# NAVAL POSTGRADUATE SCHOOL
## Monterey, California


# THESIS



THE DESIGN OF A NAVIGATOR
FOR A TESTBED AUTONOMOUS
UNDERWATER VEHICLE

by

John R. Friend

December 1989

Thesis Advisor Roberto Cristi

Inclassified

ecurity Classification of this page

## REPORT DOCUMENTATION PAGE

| | |
|---|---|
| a Report Security Classification Unclassified | 1b Restrictive Markings |
| a Security Classification Authority | 3 Distribution Availability of Report |
| b Declassification/Downgrading Schedule | Approved for public release; distribution is unlimited. |
| Performing Organization Report Number(s) | 5 Monitoring Organization Report Number(s) |
| a Name of Performing Organization Naval Postgraduate School (6b Office Symbol (If Applicable) 53) | 7a Name of Monitoring Organization Naval Postgraduate School |
| c Address (city, state, and ZIP code) Monterey, CA 93943-5000 | 7b Address (city, state, and ZIP code) Monterey, CA 93943-5000 |
| a Name of Funding/Sponsoring Organization NPS (8b Office Symbol (If Applicable)) | 9 Procurement Instrument Identification Number |
| c Address (city, state, and ZIP code) | 10 Source of Funding Numbers |

| Program Element Number | Project No | Task No | Work Unit Accession No |
|---|---|---|---|
| | | | |

1 Title (*Include Security Classification*) DESIGN OF A NAVIGATOR FOR A TESTBED AUTONOMOUS UNDERWATER VEHICLE

2 Personal Author(s) FRIEND, John R.

| 3a Type of Report | 13b Time Covered | | 14 Date of Report (*year, month,day*) | 15 Page Count |
|---|---|---|---|---|
| Master's Thesis | From | To | 1989, December, 15 | 216 |

6 Supplementary Notation The views expressed in this thesis are those of the author and do not reflect the official olicy or position of the Department of Defense or the U.S. Government.

| 7 Cosati Codes | | | 18 Subject Terms (*continue on reverse if necessary and identify by block number*) |
|---|---|---|---|
| ield | Group | Subgroup | Autonomous underwater vehicles, artificial intelligence, robotics, graphics |
| | | | |

9 Abstract (*continue on reverse if necessary and identify by block number*

Autonomous Underwater Vehicles (AUV) are a subject of study at the Naval Postgraduate School (NPS). his thesis discusses the formulation of a navigator for the Testbed AUV being constructed at NPS.

The navigator being proposed estimates the position of the vehicle using measurable dynamic arameters. The estimate is refined by an observation of position using sonar. The effects of set, drift and sonar oise are minimized using a filter. A simulation to test the AUV is also provided to test the effectiveness of the avigator through a range of noise environments. The simulation is an extension of the work of LT Hartley who odeled a more complex sonar to detect motion over the ground for a station keeping AUV.

| 0 Distribution/Availability of Abstract | 21 Abstract Security Classification |
|---|---|
| [X] unclassified/unlimited [ ] same as report [ ] DTIC users | Unclassified |

| 2a Name of Responsible Individual | 22b Telephone (*Include Area code*) | 22c Office Symbol |
|---|---|---|
| Roberto Cristi | (408) 646-2223 | Code 62Cx |

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted security classification of this page

All other editions are obsolete Unclassified

Approved for public release; distribution is unlimited.

# DESIGN OF A NAVIGATOR FOR A TESTBED AUTONOMOUS UNDERWATER VEHICLE

by

John Robert Friend
Lieutenant Commander, United States Navy
B.S., University of Virginia, 1976

Submitted in partial fulfillment of the requirements for
the degree of

**MASTER OF SCIENCE IN ELECTRICAL ENGINEERING**

from the

**NAVAL POSTGRADUATE SCHOOL**
December 1989

## ABSTRACT

Autonomous Underwater Vehicles (AUV) are a subject of study at the Naval Postgraduate School (NPS). This thesis discusses the formulation of a navigator for the Testbed AUV being constructed at NPS.

The navigator being proposed estimates the position of the vehicle using measurable dynamic parameters. The estimate is refined by an observation of position using sonar. The effects of set, drift and sonar noise are minimized using a filter. A simulation to test the AUV is also provided to test the effectiveness of the navigator through a range of noise environments. The simulation is an extension of the work of LT Hartley who modeled a more complex sonar to detect motion over the ground for a station keeping AUV.

# TABLE OF CONTENTS

## LIST OF FIGURES

## ACKNOWLEDGMENTS

I would like to express my thanks to Professor Roberto Cristi for his help during this project. I also would like to thank Professor Michael Zyda whose assistance in the software formulation and documentation of this work was invaluable. Thanks to Ray Rogers for his help and council during the entire project. Finally, I would like to thank my wife, Patricia, for her support during my two years at the Naval Postgraduate School.

# I. INTRODUCTION

## A. BACKGROUND

Since the early 1960's the Navy has been interested in Unmanned Underwater Vehicles (UUVs), which include both Autonomous (AUV) as well as Remotely Operating Vehicles (ROV). The latter are tethered to a supporting platform where an operator provides for the necessary control signals. In this case the ROV provides for a remote presence, in the sense that most of the mission planning and on-line decision making is accomplished by the operator. Increasing computer capabilities have decreased the need to tether the vehicle to provide guidance and navigational support. These advances have allowed underwater vehicles used for some missions to be autonomous.

Autonomous Underwater Vehicles (AUV) are being considered by the U. S. Navy for a variety of missions [Ref. 1]. At the Naval Postgraduate School (NPS), this interest has manifested itself in a program which has constructed one small prototype AUV with a larger Testbed Autonomous Underwater Vehicle presently under construction.

The requirement for autonomy demands that the vehicle has a navigation system capable of estimating its own position. At the present time, commercially-available inertial navigation systems exist which could provide such a vehicle with an accurate positional estimate for lengthy periods of time. However, because of cost, size, and power requirements, these systems are not suitable for at least the early prototype being

constructed by NPS. Therefore, the emphasis of the program has centered on less accurate mechanical gyros and periodic position updating by external fixes.

## B. GOALS

The goal of this thesis is to develop a system to accurately determine the position of the Testbed AUV in a test pool environment. At the same time, the development of a realistic simulation model of the sonar based on the navigation system allows for testing the techniques and concepts presented.

This thesis is divided into four parts. A background chapter which presents the vehicle design and discusses previous research on navigation systems for AUVs conducted by NPS students. This is followed by a problem statement and proposed solution which discusses modeling of the AUV environment and a mathematical discussion of position estimation. The next chapter discusses the develpment of the software simulation and presents some results of the simulation studies. The thesis concludes with a summary of simulation results and recommendations for further research.

# II BACKGROUND

## A. INTRODUCTION

The problem to be solved is to estimate the vehicle position from the integration of various sensors. In particular, gyro signals, side sonar signals, bottom tracking, depth, and speed sensors are processed by the AUV's onboard computer to derive a position estimate.

The computer memory has information of the environment, in terms of fixed material constraints (walls, rocks, shore line, etc.) and bottom map description. In particular, this thesis addresses the problem of navigating in a swimming pool, which at the present time is the environment in which the Testbed AUV will initially be tested. The information on the swimming pool stored for this purpose consists of the bottom map and the general layout of the walls of the pool.

## B. DESCRIPTION OF THE TESTBED UNDERWATER VEHICLE

The design of the vehicle under construction is based on the Swimmer Delivery Vehicle used for the transport of the U. S. Navy Special Warfare Teams. The hull shape is a flattened cylinder with a rounded bow and a tapered stern. The vehicle will be 84 inches long, 10 inches high with a 16 inch beam and a projected weight of 350 pounds. In addition to control surfaces and two propellors, it has two vertical and two horizontal thrusters for maneuvering at low speed. Figure 1 shows a drawing of the

vehicle. Locations for the sonar transducers and thrusters are annotated on the drawing.

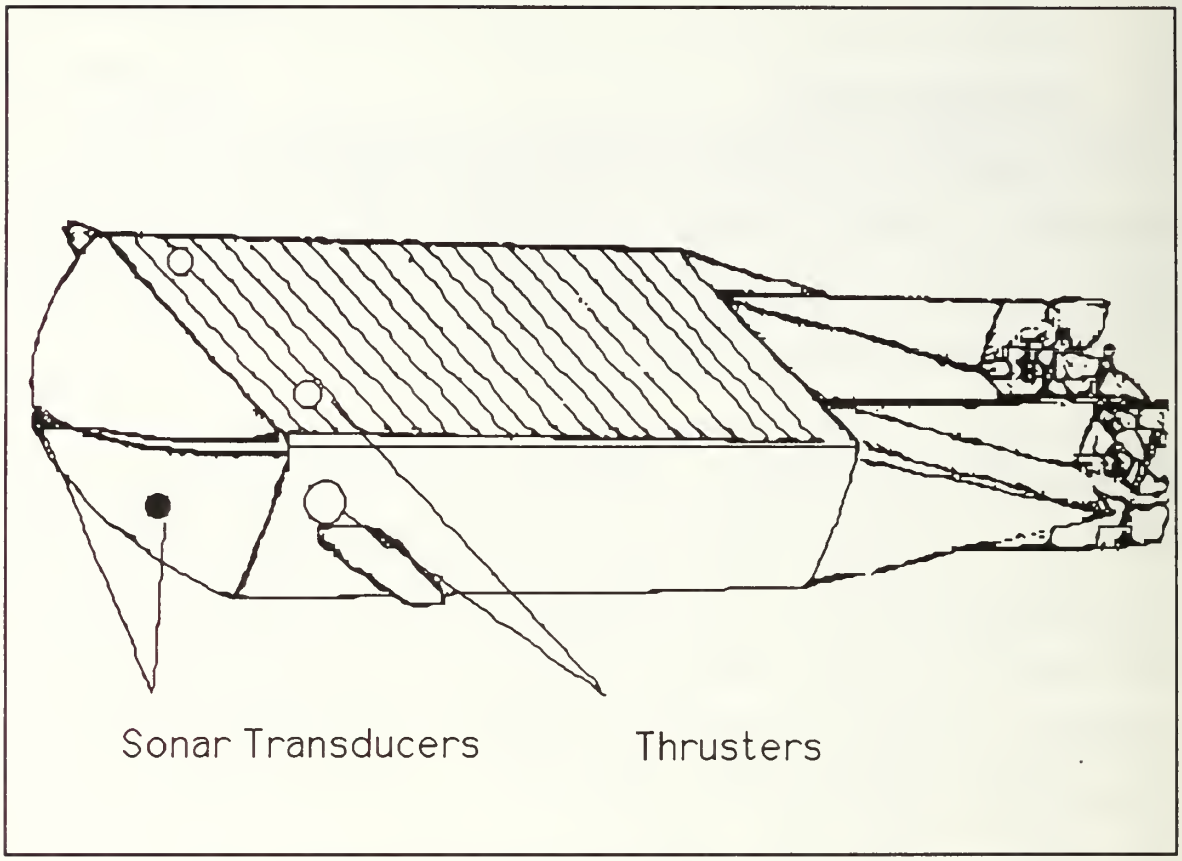


Figure 1. AUV Under Construction

The sonar and the guidance package directly affect the operation of the navigator. The sonar consists of four transducers, each one having a beam width of ten degrees. Three of these are positioned on the bow with a forty-five degree spacing, and one transducer is directed toward the pool bottom. They are to be used for obstacle avoidance and navigation and are manufactured by the Datasonics Corporation as their model PSA-900 Programmable Sonar Altimeter. The manufacturer

indicates the primary use of this sonar as a wave height measurement device, but indicates secondary uses as obstacle avoidance, surveying and depth measurement by a remotely operated vehicle [Ref. 2]. The operating characteristics of the sonar are listed in Table 1.

The directional gyroscope is a two degree of freedom system providing heading information with an accuracy of .5 degrees. A vertical gyro provides tilt angle information to the vehicle while rate gyroscopes provide pitch, and roll and yaw rate information.

## C. SURVEY OF PREVIOUS WORK

Previous research at NPS has focused on the navigation problem of the AUV. In particular Putnam [Ref. 3], proposed a conceptual design of an inertial navigation system for an autonomous submersible vehicle. Easton [Ref. 4] developed a model simulating forces acting on an underwater vehicle with the Navstar Global Positioning System and Kalman Filtering, while Hartley [Ref. 5] developed a computer simulation for the station-keeping of an autonomous submersible using bottom-tracking sonar.

This thesis is an extension of the work in Reference 5 in the sense that the navigation system of the AUV operates while the vehicle is in motion, using bottom information from the sonar system. Hartley's thesis [Ref. 5] addresses the problem of detecting AUV motion with respect to the ocean floor when the AUV was hovering over a specific area. Detecting the AUV movement with respect to the ocean floor is important when designing a control system for station keeping.

| | |
|---|---|
| OPERATING FREQUENCY | 210 KHz |
| BEAM PATTERN | 10 degrees conical |
| PULSE LENGTH | 350 microseconds |
| REPETITION RATE | User selectable 10, 1 |
| | or 0.1 pulse per second |
| RANGE | 30 meters, 1 cm resolution |
| | or 300 meters, 10 cm resolution |
| ACCURACY | +/- .25% of full scale |
| RANGE OUTPUT | 0 to 10 volts DC |
| | or 1 to 11 KHz 50% duty cycle |
| TIME-VARYING GAIN | 60 db compensation |
| CONTROL | for loss of 20 log(2R) + 2R |
| EXTERNAL KEY | 0 to 5 volts TTL compatible |
| ERROR FLAG | 0 to 5 volts, TTL signal |
| | on missed echo or over range |
| DEPTH (PRESSURE) OUTPUT | 0 to 5 volts DC or 0 to 10 khz |
| | representing zero to full scale |
| OPERATING DEPTH | 200 meters |
| DIMENSIONS | 4 in OD x 10.5 in long |
| WEIGHT | 8 lbs in air, 5 lbs in water |
| POWER REQUIRE' 1ENT | 15 to 28 volts @ 100 ma |

Table 1. Sonar Specifications [Ref. 2]

Hartley proposed to solve this motion detection problem by scanning the ocean floor with the bottom-tracking sonar, constructing a contour

map of the bottom with the sonar return data, and then comparing this scan with another similarly generated scan. The best match of the two contour maps would then be used to determine the direction and distance of vehicle motion. Reference 5 develops a simulation program on a SILICON GRAPHICS IRIS 2400 workstation, using the WESMAR SS265 sonar as a model. The simulation has three different displays: an initialization screen allows various vehicle parameters and location to be set prior to simulation start, a scanning screen which provides a depiction of sonar video display and allows the operator to alter various scanning parameters and observe the change on the simulation, and a storage array display screen which shows the last two terrain maps stored. The sonar chart and display are generated from an actual digital topographical map that is loaded at program start.

## D. SUMMARY

This chapter describes the structure of the Testbed AUV, the characteristics of the gyro and sonar systems, and includes a discussion of previous navigation systems research conducted at NPS for the AUV project. In particular the thesis work of Hartley [Ref. 5] is discussed since this thesis is an extension of his efforts.

The next chapter addresses the problems encountered in estimating the position of the Testbed AUV. A mathematical algorithm for determining the vehicle position based on a simple kinematic model is developed. In addition, a mathematical model of a simulation is presented which will be used to test the navigation algorithm.

# III. PROBLEM STATEMENT AND PROPOSED SOLUTION

## A. INTRODUCTION

For the vehicle to be autonomous and make decisions concerning path planning, it has to be able to accurately determine its own position. The vehicle under development has limited capabilities in terms of sonar and inertial navigation accuracy. However this is not a major drawback for navigation within well charted areas.

This chapter discusses the proposed method of position estimation, regression analysis, and data filtering which will be used by the AUV navigator. Also included is a discussion of the AUV sonar simulation which will be used to evaluate the proposed navigator.

## B. OVERVIEW OF NAVIGATOR OPERATION

This section provides an overview of the operation of the navigator. A flow diagram of the process is shown in Figure 2. The vehicle's speed, depth, roll, pitch, heading and sonar information are recorded at 2 second intervals. The vehicle uses the current angle, heading and speed information to estimate its position at the next sample time. At the next interval an observation of position is made using a comparison of sonar information with prestored terrain information. A quality factor is computed for each observation and this quality factor is used in a filtering process to weight the observation. The estimated position is updated from the previous estimate on the basis of the observation and its quality of fit.

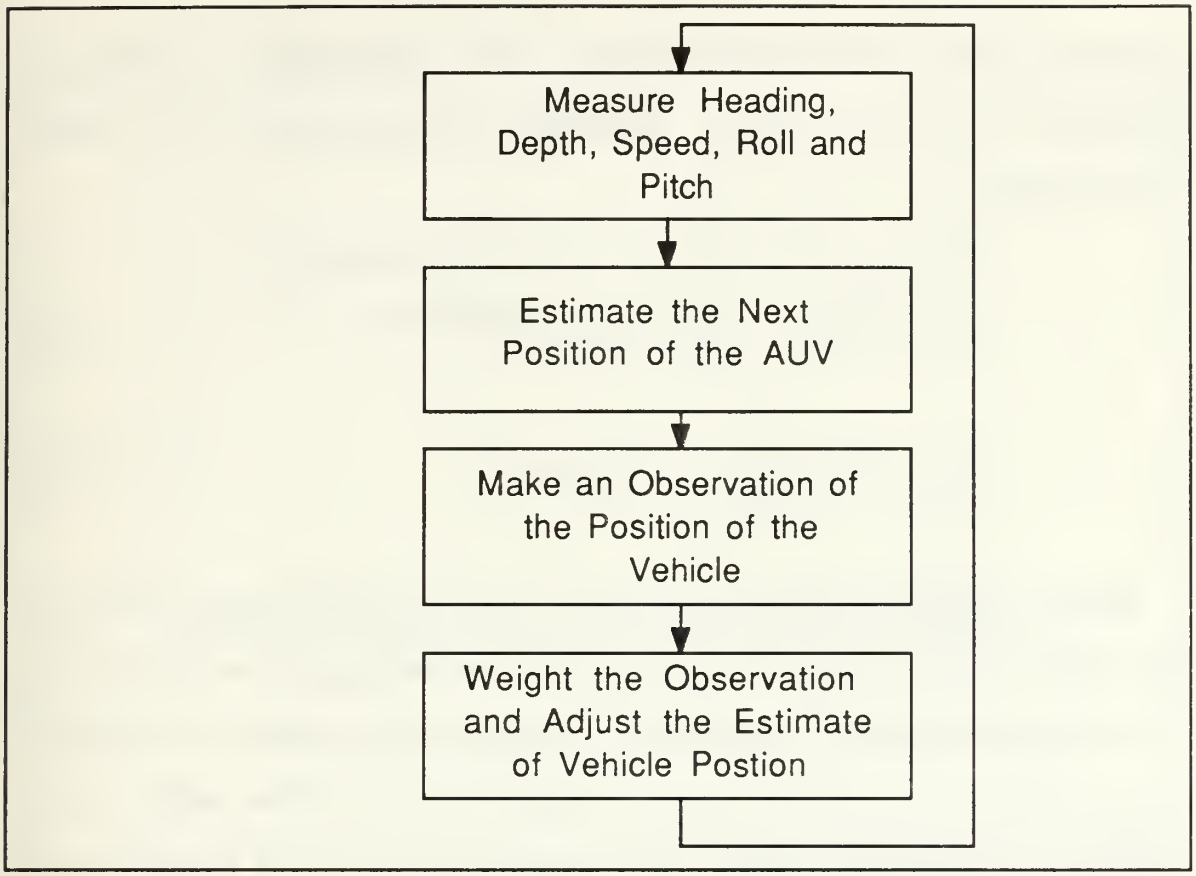


Figure 2. System Flowchart

## C. MATHEMATICAL MODEL OF THE VEHICLE DYNAMICS

This section presents a dynamic model of the AUV moving in the horizontal plane. As the AUV travels through the water with a desired heading and speed, external forces such as currents or other hydrodynamic factors cause the vehicle to deviate from the desired track. We can view this as a perturbation in velocity (drift) or heading (set). Following a similar approach as in Reference 4, these deviations can be modeled as a single velocity vector added to the base velocity. The

resultant is the true velocity vector. This relationship is shown in Equations 3.1 and 3.2, where $\Psi$ is the angle of the set and $\gamma$ is the vehicle heading angle.

$$\dot{X}_T = \text{Vel} \cos(\gamma) + \text{Drift} \cos(\Psi) \tag{3.1}$$

$$\dot{Y}_T = \text{Vel} \sin(\gamma) + \text{Drift} \sin(\Psi) \tag{3.2}$$

Figure 3 illustrates the total dynamic model of all the forces acting on the vehicle where **w** is the forcing function caused by set and drift and **u** is a forcing function of velocities generated by the transfer function **G** from ordered course and speed changes and $\alpha$ is the angle of the composite true velocity vector. The transfer function G is nonlinear and varies with vehicle speed. However linear approximations [Ref. 6: p 45] of the vehicle dynamics can be made for specific speeds. This model can be represented in state space as summarized in Equations 3.3 through 3.8 and Table 2.

In the estimation problem a simpler state space model is used (Equations 3.3 and 3.4). It is based on the fact that the trajectory of the AUV is, in general, smooth and the position of the vehicle within the sampling interval can be predicted assuming constant velocity.

$$\underline{x}_{k+1} = \Phi \underline{x}_k + \Gamma_1 \underline{u}_k + \Gamma_2 \underline{w}_k \tag{3.3}$$

$$\underline{z} = H \underline{x}_k + \underline{v}_k \tag{3.4}$$

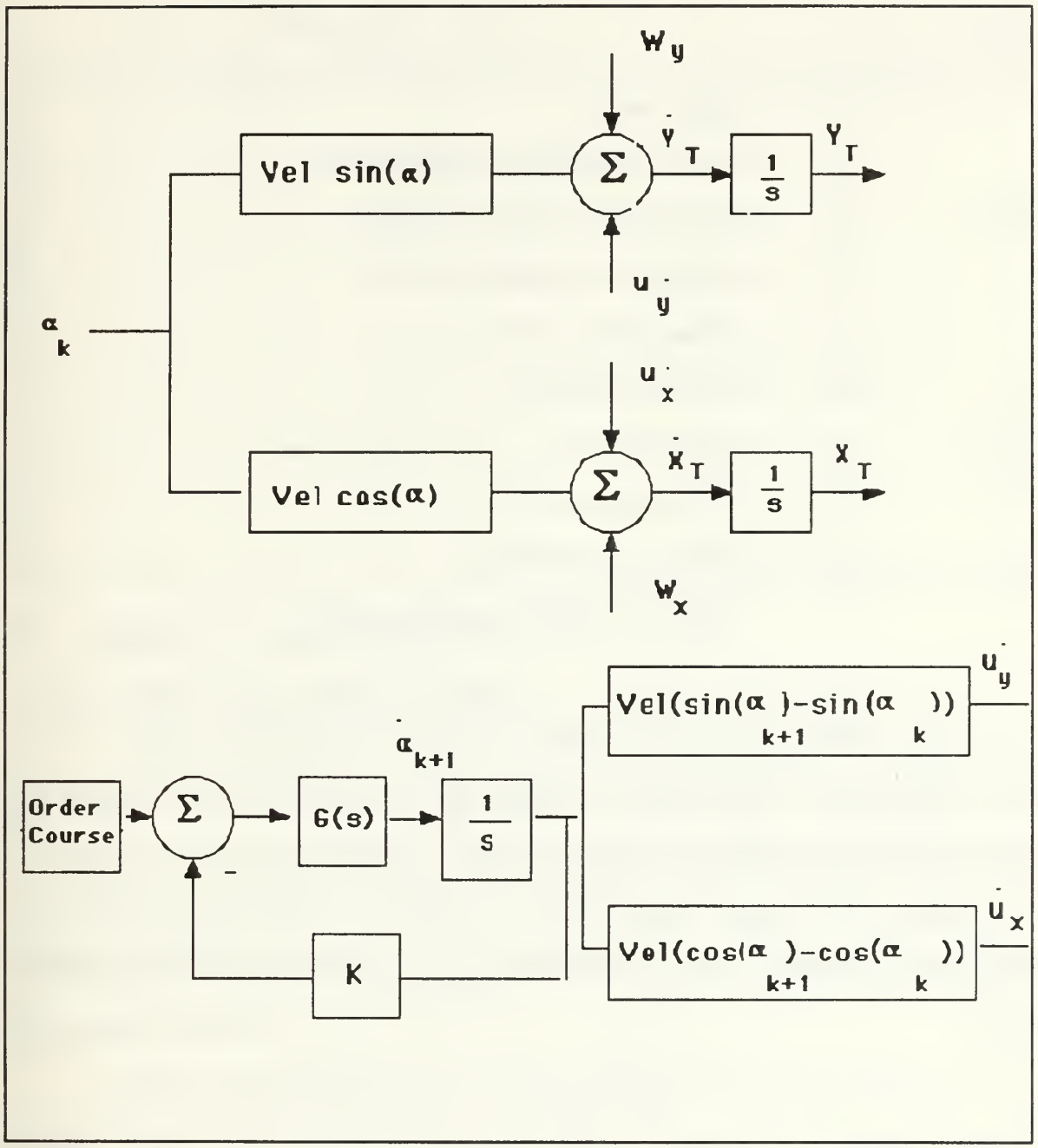


Figure 3. Dynamic Model

| | |
|---|---|
| $\underline{x}$ | = State vector |
| $\underline{u}_k$ | = Vector of applied forcing functions |
| $\underline{w}_k$ | = Set and drift forcing functions |
| $\underline{z}$ | = Position observation vector |
| $\underline{v}$ | = Measurement noise vector |
| $\Phi$ | = Transition matrix |
| $\Gamma_1$ | = u input matrix |
| $\Gamma_2$ | = Noise input matrix |
| $\underline{H}$ | = Observation matrix |

Table 2. Symbol Identification

This is expressed by the $\Phi$ and $\Gamma$ matrices in Equations 3.6, 3.7 and 3.8. The term $u_k$ refers to the velocity perturbations $\dot{u}_x$, $\dot{u}_y$ which in principle, can be computed from the lower flowchart shown in Figure 3 and which result from course and speed changes. However, to make the calculations simple, these perturbations are modeled as random white noise.

$$\underline{x} = \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix} \tag{3.5}$$

$$\Phi = \begin{bmatrix} 1 & T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

$$\Gamma_1 = \begin{bmatrix} T & 0 \\ 1 & 0 \\ 0 & T \\ 0 & 1 \end{bmatrix} \tag{3.7}$$

$$\Gamma_2 = \begin{bmatrix} T & 0 \\ 1 & 0 \\ 0 & T \\ 0 & 1 \end{bmatrix} \tag{3.8}$$

## D. POSITION OBSERVATION

### 1. Sonar data transformation

The observation vector **z** is derived from sonar observations of the terrain and consists only of position coordinates. The method of obtaining these observations is similar to that used in Reference 2. The beam tip depth of the sonar is transformed into x_y coordinates referenced to the pool using Equations 3.9, 3.10 and 3.11 and graphically depicted in Figure 4.

The sonar information is contained in the beam length and its angular coordinates, $\Psi_b$ and $\Theta_b$. In order to estimate the vehicle position in world coordinates it is necessary to transform this information into a vector in the Cartesian reference frame of the pool. Initially the position of the beam with respect to the coordinate frame of the vehicle is

determined using Equation 3.9 where $\Theta_b$ = Tilt Angle and $\Psi_b$ = Sonar Beam Heading.

$$P_b = \begin{bmatrix} \text{beam-length} * \cos\Theta_b * \cos\Psi_b \\ \text{beam-length} * \cos\Theta_b * \cos x\Psi_b \\ \text{-beam-length} * \sin\Theta_b \\ 1 \end{bmatrix} \tag{3.9}$$

This vector is then converted into a world coordinate frame using the transformation matrix in Equation 3.10 where c = cos, s = sin, $\Psi_a$ = AUV Heading (Azimuth), $\Theta_a$ = AUV Dive Angle (Elevation) and $\phi_a$ = AUV Roll Angle.

$$^{w}A_b = \begin{bmatrix} c\Psi_a c\Theta_a & c\Psi_a s\Theta_a s\phi_a - s\Psi_a c\phi_a & c\Psi_a s\Theta_a s\phi_a + s\Psi_a c\phi_a & x_a \\ s\Psi_a c\Theta_a & c\Psi_a c\phi_a + s\Psi_a s\Theta_a s\phi_a & s\Psi_a s\Theta_a c\phi_a - c\Psi_a s\phi_a & y_a \\ -s\Theta_a & c\Theta_a c\phi_a & c\Theta_a c\phi & z_a \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

The elements $x_a$, $y_a$ and $z_a$ in this matrix refer to the location of the center of the AUV sonar scan mechanism in pool coordinates.

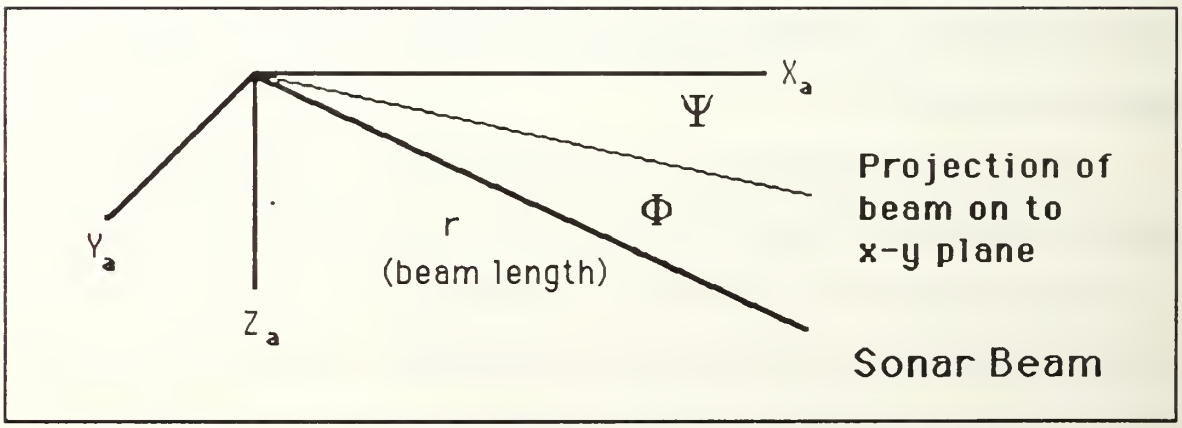


Figure 4. Sonar Beam Tip Coordinate Transformation

From Equations 3.9 and 3.10, the vector $P_W$, which represents the beam tip in Cartesian pool coordinates, can be calculated as

$$P_W = {}^{W}A_b * P_b. \tag{3.11}$$

This entire transformation is represented by the following three equations, where ($x\_coord_w$, $y\_coord_w$ and $z\_coord_w$) indicates the beam tip postion in world coordinates, while ($x\_coord_a$, $y\_coord_a$ and $z\_coord_a$) represents the location (in world coordinates) of the vehicle.

$$\begin{aligned} x\text{-}coord_W = x\text{-}coord_a &+ (\cos\Psi_a \cos\Theta_a \textit{beam-length} \cos\Theta_b \cos\Psi_b) + \\ &((\textit{beam-length}\ \cos\Theta_b \sin\Psi_b) \times ((\cos\Psi_a \sin\Theta_a \sin\phi_a) - (\sin\Psi_a \cos\phi_a)) \\ &((\textit{beam-length}\ \sin\Theta_b) \times ((\cos\Psi_a \sin\Theta_a \sin\phi_a) + (\sin\Psi_a \cos\phi_a))) \end{aligned} \tag{3.12}$$

$$\begin{aligned} y\text{-}coord_W = y\text{-}coord_a &+ (\sin\Psi_a \cos\Theta_a \textit{beam-length} \cos\Theta_b \cos\Psi_b) + \\ &((\textit{beam-length} \cos\Theta_b \sin\Psi_b) \times ((\cos\Psi_a \cos\phi_a) - (\sin\Psi_a \sin\Theta_a \cos\phi_a))) - \\ &((\textit{beam-length}\ \sin\Theta_b) \times ((\sin\Psi_a \sin\Theta_a \cos\phi_a) + (\cos\Psi_a \sin\phi_a))) \end{aligned} \tag{3.13}$$

$$\begin{aligned} z\text{-}coord_W = z\text{-}coord_a &+ (\sin\Theta_a \textit{beam-length} \cos\Theta_b \cos\Psi_b) + \\ &(\cos\Theta_a \sin\phi_a \textit{beam-length} \cos\Theta_b \sin\Psi_b) - \\ &(\cos\Theta_a \cos\phi_a \textit{beam-length}\ \sin\Theta_a). \end{aligned} \tag{3.14}$$

The depth values measured by the sonars in base coordinates are then stored in a 300 by 300 x_y coordinate array in which the vehicle position is at (150, 150), as depth beneath the keel values. The array is then compared to the bottom terrain map of the pool residing in memory to determine the position of the AUV. A drawing of the vehicle, sonar beams and depth array is shown in Figure 5. The four squares on the array that are filled in represent the x_y positions of the beam tips. The lighter shading of three of the filled-in squares represents sonar returns from the pool side walls while the darker square represents the return from the AUV fathometer. In this figure, darker shading represents deeper beam tip depth.

### 2. Array to Terrain Map Comparison

The position of the vehicle is estimated iteratively from the measurements and the analytical model of the vehicle. At each iteration the sonar depth array is centered on the AUV estimated position on the terrain bottom map. If the array values match the terrain map in position and depth there is no error in the initial estimate. If the match is not exact the array is shifted in position over the terrain map in increments of a tenth of a meter, over a ten square meter area around the initial position to try and find a better match. After the array has moved to a better match, an additional search is made around the initial observation to ensure the best possible observation has been made.

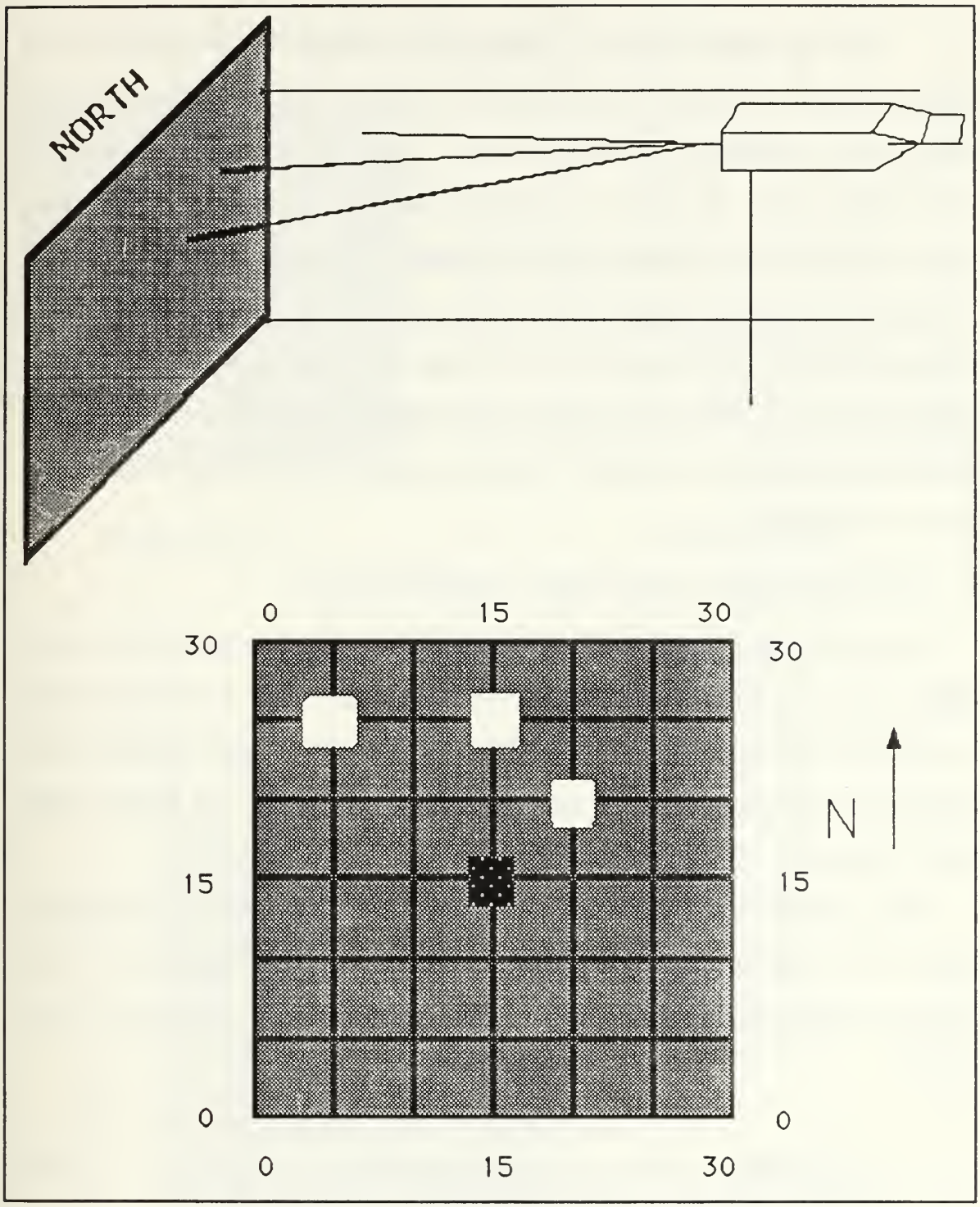


Figure 5. Sonar Array Data Storage

The position over the terrain map where the array is finally centered is the observed position of the AUV. Figure 6 illustrates the steps in the comparison procedure where the sonar array is compared to the bottom map. In Figure 6 the best match is found when the three sonar returns (white squares) are compared to the terrain map values in the upper right-hand corner. This places the observed position of the vehicle, which is in the center of the sonar array, in the upper right hand corner of the terrain bottom map. During the comparison procedure, the terrain map values are modified to take into account the depth of the AUV prior to comparison.

## E. DETERMINING THE BEST OBSERVATION

The grid search method [Refs. 5 and 7] is used to compare the sonar depth array measured by the AUV and the terrain map. In this approach a criterion function, $\Omega$, is used which is the sum of the squared error between the depth values of the sonar depth array and the terrain map. Only those cells which contain data are used.

This is shown in Equation 3.15 where n is the number of overlapping cells $D_1$ is the sonar array, $D_2$ the terrain map, and $(\Delta x, \Delta y)$ is the unknown movement of the AUV with respect to the estimate of the model.

$$\Omega = \frac{1}{n} \sum (D_1(x_i,y_i) - D_2(x_i + \Delta x, y_i + \Delta y))^2 \qquad (3.15)$$

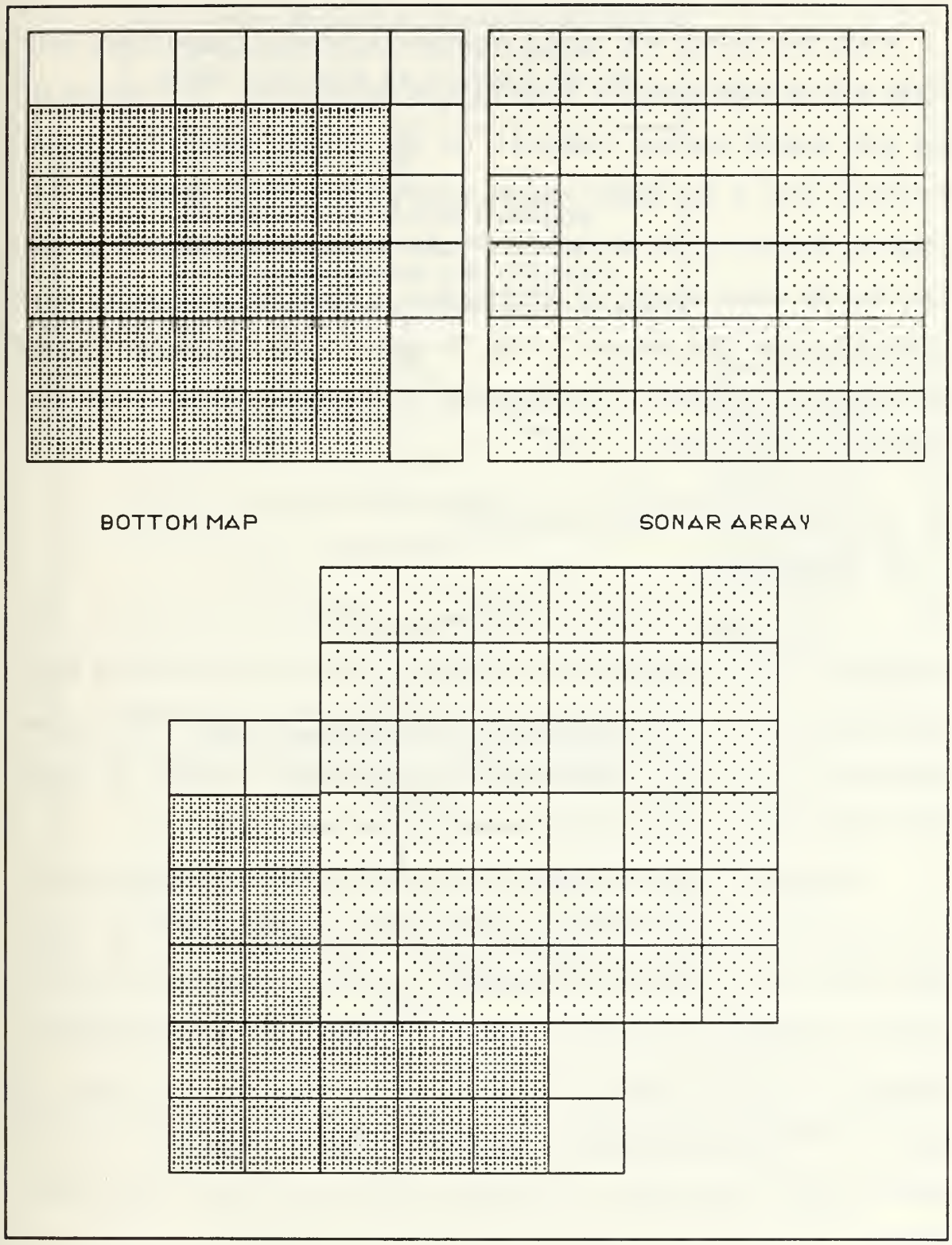


Figure 6. Sonar Array to Bottom Map Comparison

When this method was used in References 5 and 7, comparisons were made only at eight positions around the initial estimate. This version of the grid search method compares for the correct match for 10,000 positions over a ten meter square area as described earlier. In this program, because of the limited sonar information, more comparisons are used but the actual number of computations is about the same as the ones carried out in References 5 and 7 where more extensive sensor information is available. A flowchart of this process is shown in Figure 7 [Ref. 5].

## F. METHOD OF DATA FILTERING

### 1. Introduction

The amount of sonar information that can be used by the navigator of the testbed AUV is small because of the limited scan capabilities of the sonar being used. The amount of noise that can be introduced into such a sonar could be quite high because the wide beamwidth (10°) and interference between the transducers.

In spite of these limitations, we can still use the sonar information to measure and correct for navigational errors. To ensure that the sonar noise does not degrade the estimation correction process, a filter is needed to minimize the errors caused by the noise. A Kalman filter was chosen for this task because it is an iterative method that can obtain an optimal estimate of a variable in a noisy environment. The filter uses measurements of the plant to recursively refine the estimate of the plant's state.

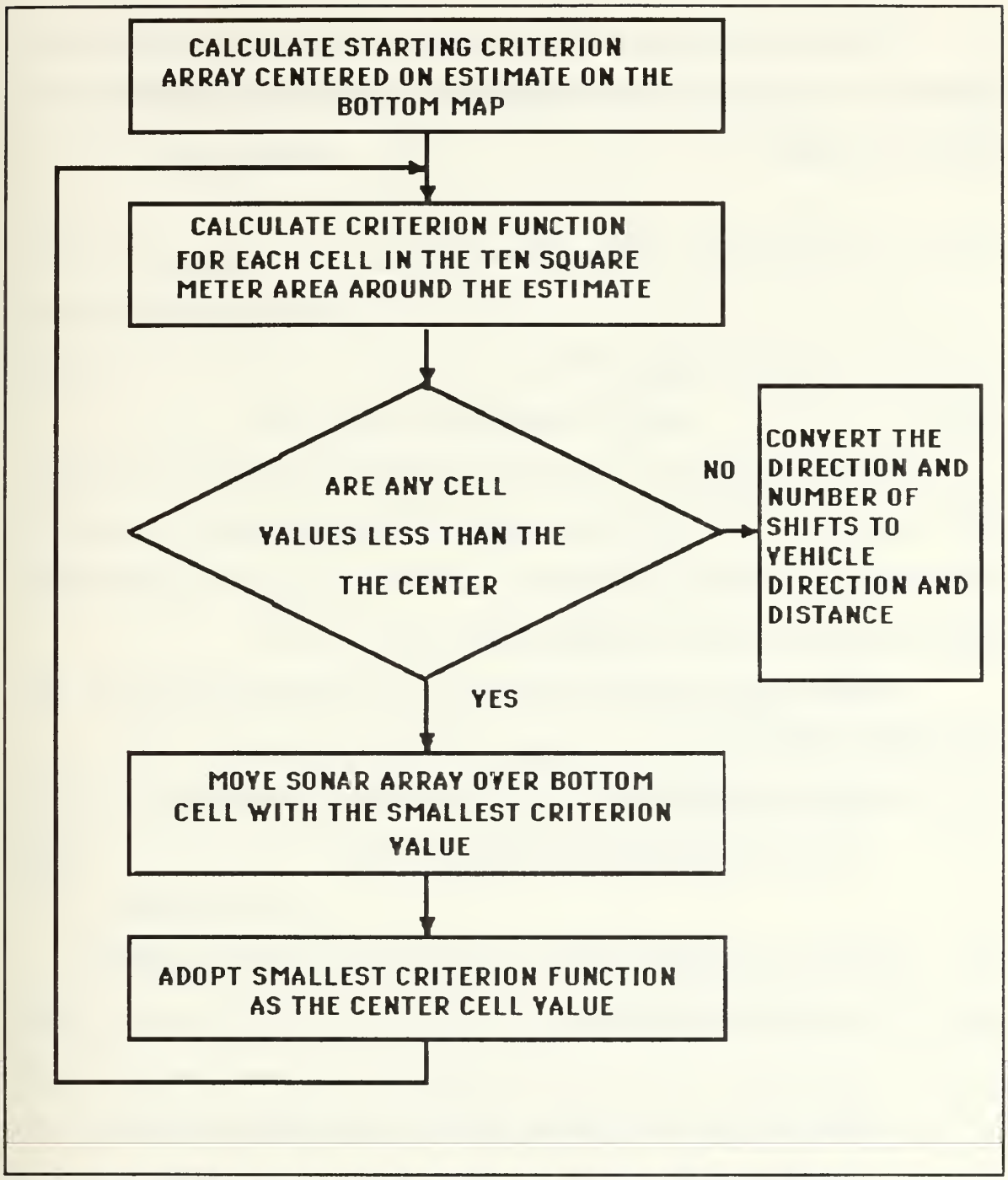


Figure 7. Least Squares Observation Flowchart

When the system and measurement models are linear as in Equations 3.3 and 3.4, the state update equations shown in Equations 3.16 and 3.17 are valid.

$$\underline{x}_{k/k} = \underline{x}_{k/k-1} + G(\underline{z}_k - \underline{H}\underline{x}_{k/k-1}) \tag{3.16}$$

$$\underline{x}_{k+1/k} = \Phi\, \underline{x}_{k/k} + \Gamma u_k \tag{3.17}$$

In this equation G is the vector containing the optimum estimation gains, $\underline{x}_{k/k-1}$ is the predicted state at time kT based on known observations and forcing functions at (k-1)T with $\underline{x}_{k/k}$ the updated state based on the last position observation $\underline{z}_k$.

When using the Kalman equations the following assumptions are made [Refs. 4 and 8]:

- $w_k$ is a random forcing function which is zero mean, uncorrelated and has a covariance $Q_k$.

- The measurement noise $v_k$ is zero mean and uncorrelated with a covariance of $P_k$.

- The randc n forcing function and measurement noise are uncorrelated.

- The random forcing function and the initial states are uncorrelated.

- The measurement noise and the initial states are uncorrelated.

The gains which satisfy these conditions are recursively updated by the following equations.

$$G_k = P_{k/k-1}\, H^T[H_k P_{k/k-1} H^T + R]^{-1} \tag{3.18}$$

$$P_{k/k} = [I - G_k H_k] P_{k/k-1} \tag{3.19}$$

$$P_{k+1/1} = \Phi P_{k/k} \Phi^T + \Gamma_2 Q \Gamma^T \tag{3.20}$$

$P_{k/k}$ is the error covariance matrix at time kT. The components of the Kalman filter equation are selected to minimize the effects of sonar noise.

**2. Selection of Q**

Equation 3.3 is the state space representation of the system. The forcing function for Set and Drift, $\underline{w}_k$, is assumed to be white Gaussian noise with zero mean with components in the x and y directions. Q is in this case the covariance matrix for the random forcing function of the Set and Drift velocity components.

$$Q = \begin{bmatrix} \sigma_{wx}^2 & 0 \\ 0 & \sigma_{wy}^2 \end{bmatrix} \tag{3.21}$$

Using the procedure outlined in Reference 3 with some modification to take into account the sonar system and operating area configuration, $\sigma_{wx}^2$ and $\sigma_{wy}^2$ were calculated as $\sigma_{wx}^2 = 0.5$ knots and $\sigma_{wy}^2 = 1.0$ knot.

**3. Selection of R.**

R is the covariance matrix for the measurement noise vector, expressing the uncertainty contained in the measuring of the two parameters x and y. The noise is assumed to be uncorrelated, white Gaussian with zero mean.

$$R = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix} \tag{3.22}$$

The quantities $\sigma_x^2$ and $\sigma_y^2$ are formulated based on the the criterion function computed in the least squares observation, the number of sonar returns or cells in the sonar array, the arrangement of the sonar cells in the sonar array and the location of the AUV. R is evaluated after each observation. The weighting of the values of R were derived from trial-and-error measurements. However the covariance of each component increases when the criterion function increases, as the number of sonar cells decrease and if the sonar conditions in a particular area are known to be poor. The covariance for a particular component is also increased if the arrangement of cells in the sonar array is such that information for that component is lessened.

**4. Selection of H and Overview of Filtering Process.**

H is the observation matrix and in this case it is the identity matrix because no transformation of the observed coordinates is needed. Figure 8 shows the overall process of filtering.

## G. SUMMARY

This chapter has presented a detailed discussion of the mathematical models being used for terrain scanning, regression analysis and data filtering used by the AUV navigator. The next chapter will present a description of the AUV sonar and navigation simulation, an operating manual for the simulation and simulation results.

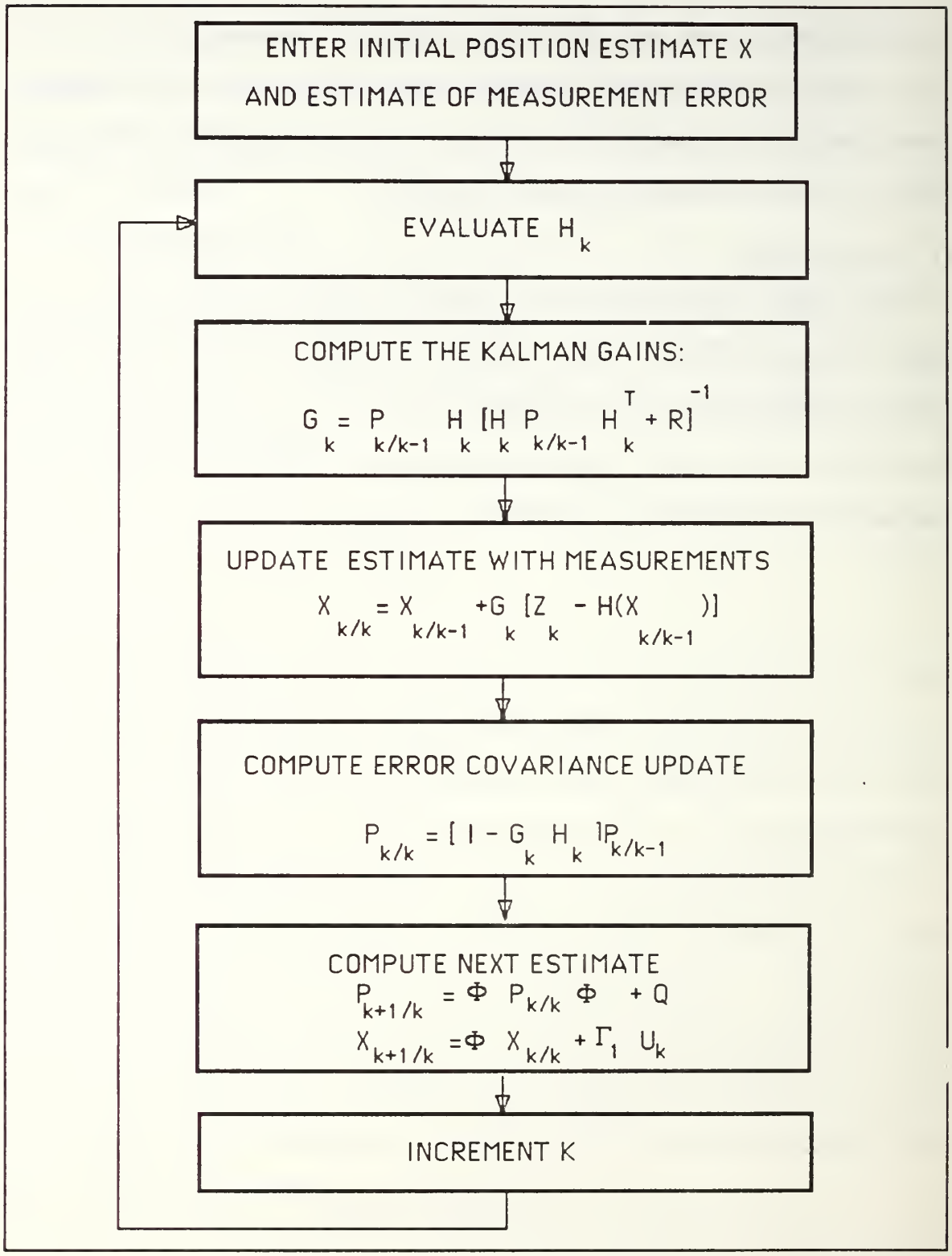


Figure 8. Kalman Filter Recursive Loop

# IV SIMULATION PERFORMANCE

## A. INTRODUCTION

In Chapter 3, AUV position estimation, data filtering and regression analysis were discussed. To evaluate these algorithms it is necessary to test them either in an actual vehicle or by an accurate computer simulation. The software which implemented these algorithms is designed to allow testing of this approach using computer simulation and, after removal of the graphics and simulation functions and installation of interface modules, testing of the navigator in the Testbed AUV. As previously discussed, the computer simulation is implemented on an IRIS GTX workstation using the C programming language. A discussion of which interface functions are required to be modified to allow operation in the testbed AUV is included in this chapter.

## B. PROGRAM DESCRIPTION

### 1. Bottom Contour Model

The Testbed AUV will be tested in the Naval Postgraduate School swimming pool. The bottom contour map used for the regression analysis and the sonar simulation will be the swimming pool contour. This contour was obtained from a blueprint of the pool provided by the NPS public works department. The dimensions of the pool are 35 meters by 20 meters. The pool coordinates are in tenths of meters with the origin at the left corner of the shallow end as shown in Figure 9. As the AUV moves from the shallow to the deep end, the y coordinates become more

negative. This was done to provide a right handed coordinate system. Equations 4.1, 4.2 and 4.3 below are used to approximate the contour of the pool.

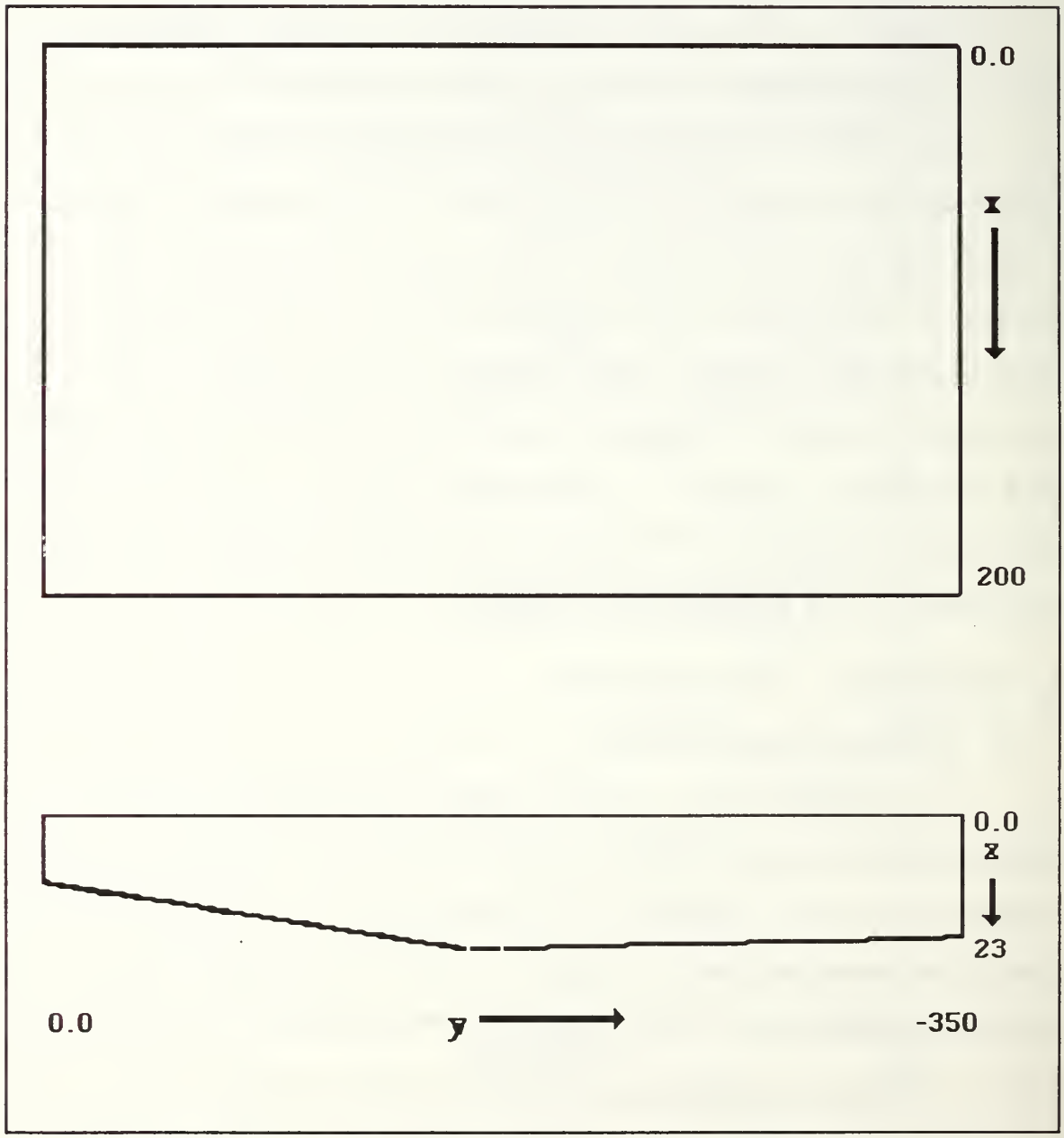


Figure 9. Diagram of NPS Pool

$$z = -.088y + 10.62 \qquad (0 < y \leq -156) \qquad (4.1)$$

$$z = -.07693y + 12.327 \qquad (-156 < y \leq -175) \qquad (4.2)$$

$$z = 1.7391E\text{-}2y + 28.96 \qquad (-175 < y \leq -351) \qquad (4.3)$$

The depth values for each contour point are calculated at the start of the program and loaded into a bottom data depth array. The depth array can be as large as 354 x 354 without requiring modification. This allows flexibility if the vehicle is tested in something other than the NPS swimming pool.

### 2. Data Analysis and Filtering Modules

Six modules perform data analysis, filtering and storage. The first is **Obs_filter.c** which performs the position observation using grid search least squares analysis and Kalman filtering of the results as discussed in Chapter 3. **Get_estimate.c** estimates the position of the vehicle based on vehicle parameters. **Store_data.c** converts the sonar return data from polar coordinates to Cartesian coordinates. The sonar data is then stored. The scan data is maintained in case some future generation of vehicle uses scan comparison for detection of vehicle motion. **Init.arrays.c** initializes the two sonar scan arrays. **Reset_depth_array.c** initializes array pointer functions. **Up_arrays.c** keeps track of which sonar scan array is active.

### 3. Video Display Modules

Nine modules are used to create various portions of the video display. These are the modules which will be removed when the program is used as the navigator in the vehicle. The first module provides an x-y Cartesian coordinate sonar array display. **Disp_estimate.c** provides a numeric output display of the best estimate of vehicle position. **Make_arrows.c** creates the direction and velocity arrows for the AUV course and speed, AUV course and speed over the ground, and the ocean current set and drift. The bottom contour of the pool is visually represented by **Make_chart.c**. **Make_depth.c** creates the depth slide bar that allows the program user to select the initial depth of the AUV when the program is being initialized. **Make_eplus.c** provides a plus sign on the chart for the estimated position of the AUV while **Make_plus.c** makes a plus sign for the actual position when the program is being initialized. **Make_inst.c** provides instructions on the AUV parameter initialization screen while **Make_readout.c** is used to generate the instrument readout of the vehicle parameters.

### 4. Data Control Modules

There are five data control modules. **Control_sweep.c** controls which sonar transducer is being sampled. **Load_data.c** constructs the terrain bottom map. **Scan_ctrl.c** determines when the scan is complete. **Main_sonar.c** is the main program which calls all functions and routines. **Int_arrays.c** initializes the sonar scan arrays.

### 5. Simulation and Miscellaneous Modules

**Get_posit.c** calculates the actual position of the AUV. **Get_sets.c** controls the program in the initialization mode. **Read_ctrls.c** reads operator inputs when the sonar is scanning and **Read_sets.c** reads operator inputs during program initialization. Both of the interface modules will require modification when the program is used as a navigator in the Testbed AUV. At the end of each program loop all boolean flags are reset by **Reset_flags.c**.

## C. SONAR SIMULATION

As in Reference 4, and as stated in the beam tip coordinate Equations 3.15, 3.16 and 3.17, a polar beam angle and length can be converted into Cartesian coordinates and used effectively in the bottom scanning model. In scanning the terrain, the sonar's beam length is incremented one coordinate unit at a time. After each increase in beam length, the x, y and z coordinates are calculated and the depth of the beam is compared to the bottom terrain map to determine if the beam is in contact with the bottom. If it is, the beam length is reduced by one half of a coordinate unit to more closely approximate the true beam length. The length is then recorded and the next sonar transducer is selected for simulation. Figure 10 shows a flowchart of the sonar simulation.

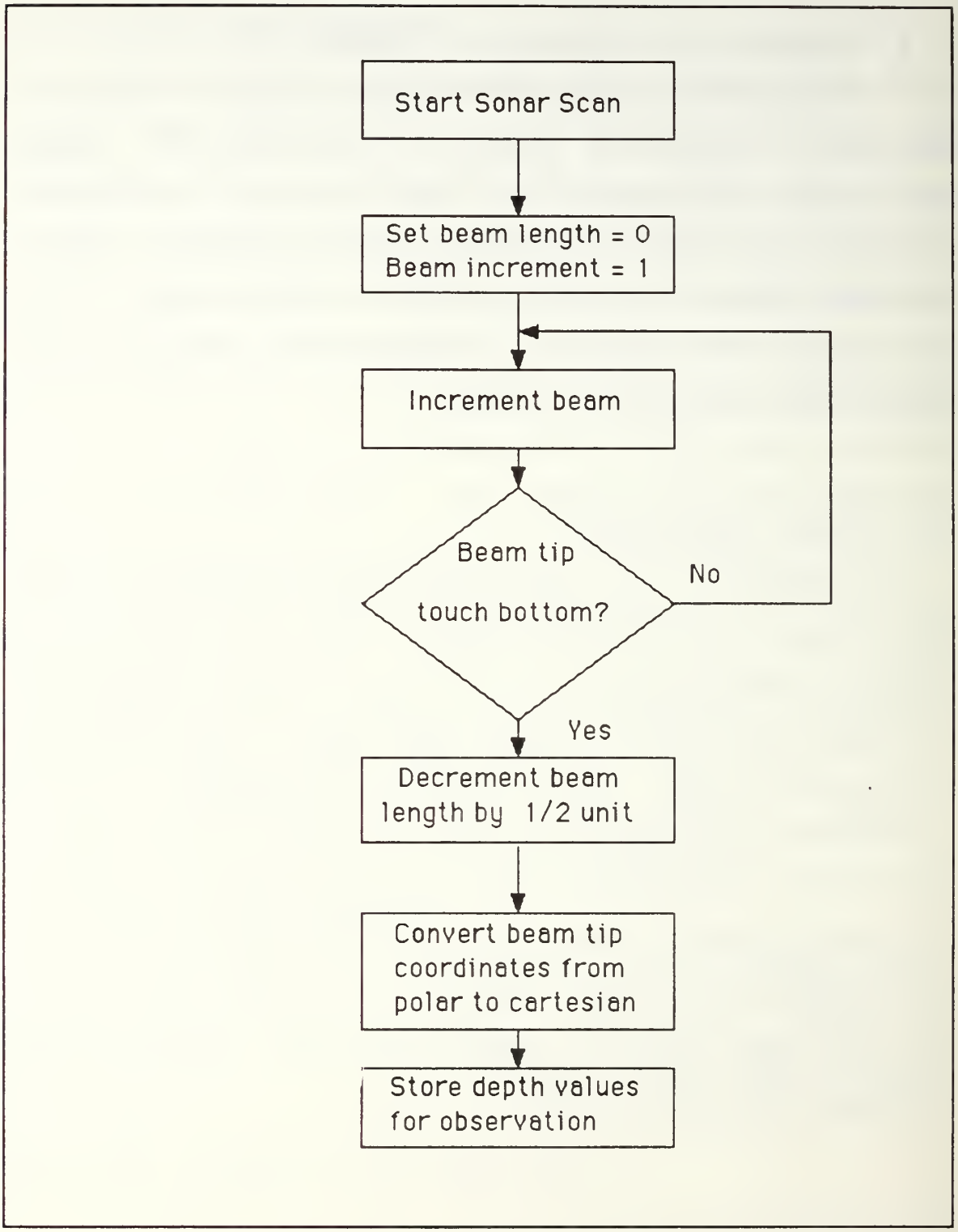


Figure 10. Sonar Simulation Flowchart

## D. USER'S MANUAL

The AUV navigation simulation consists of two parts, initialization and scanning/position Estimation. The simulation can be started by entering **nav** or **nav_file** at the IRIS GTX graphics workstation console. Typing nav starts the simulation without saving simulation data, while nav_file saves simulation data in a file called nav_data which may be reviewed and printed later. First to be displayed is the initialization screen which is shown in Figure 11. AUV course, speed, dive angle, location, ocean set and drift and sonar noise can be entered into the simulation. The initial position of the AUV may be set by placing the cursor at the desired location over the ocean floor contour chart and pressing the left mouse button. The depth bar symbolizes the depth of the water at the current AUV location, so to set the initial AUV depth at half way to the bottom, place the cursor in the middle of the bar. The AUV speed and the ocean current magnitude can be adjusted in an identical fashion by placing the cursor over the AUV speed or ocean current drift bar and pressing the left button. By turning the appropriate dials on the dial box, the AUV course, dive angle and ocean current direction may be set. The scanning mode may be set at this point also. This simulator has three modes: just scanning with no sonar return data being stored, scan with no fathometer data being stored, and a complete scan with the return data being stored. Select the scan mode by selecting scan 0, 1 or 2 respectively from the pop-up menu selection. Four levels of noise (0, 1, 5 or 10 percent deviation of beam length) can be introduced into the sonar simulation by using the menu. The scanning process can be started by selecting

*commence scan* from the pull down menu. The scan/position estimation screen is shown in Figure 12. The actual position of the vehicle is denoted on the chart by an orange dot with three arrows originating from it. These arrows represent the AUV's heading and speed (green), set and drift (yellow) and course and speed over the ground (red). The arrows are updated as the vehicle position is displayed by a plus on the chart. Two orange rings are centered on actual position with a three and fifteen meter radii to assist in evaluating the estimated error. A numeric readout of actual and estimated position is also given in the instrument section. In the upper right corner of the screen there is a Cartesian sonar array display. The display is oriented so that north (the deep end) is at the top of the display. The center coordinate of the sonar display is the actual AUV position. By making the appropriate menu selection the simulation can be directed to either stop after each scan or be free running. Any of the three sonar modes can be selected, during the scanning process. All other vehicle parameters may be adjusted in the scanning mode with the exception of depth. There is a lag between ordering a vehicle parameter change and achievement of the order because of the dynamics of the vehicle. The user is able to return the simulation to the initialization mode by a menu selection or by allowing the vehicle to run aground. When the vehicle does run aground, a red warning will appear in the instruction section of the program and the simulation will be immediately returned to the initialization mode to allow the vehicle to be repositioned. The program is exited from the scan/position estimation mode by selecting quit on the menu.

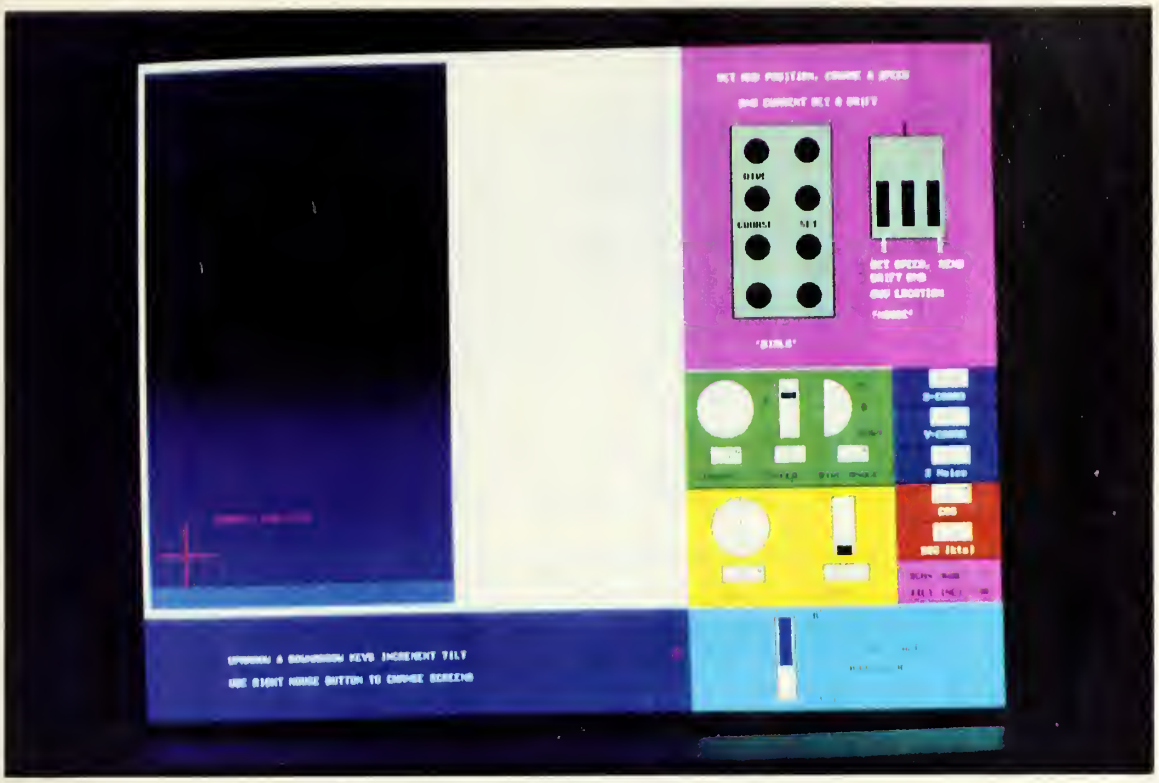

Figure 11. Initialization Screen

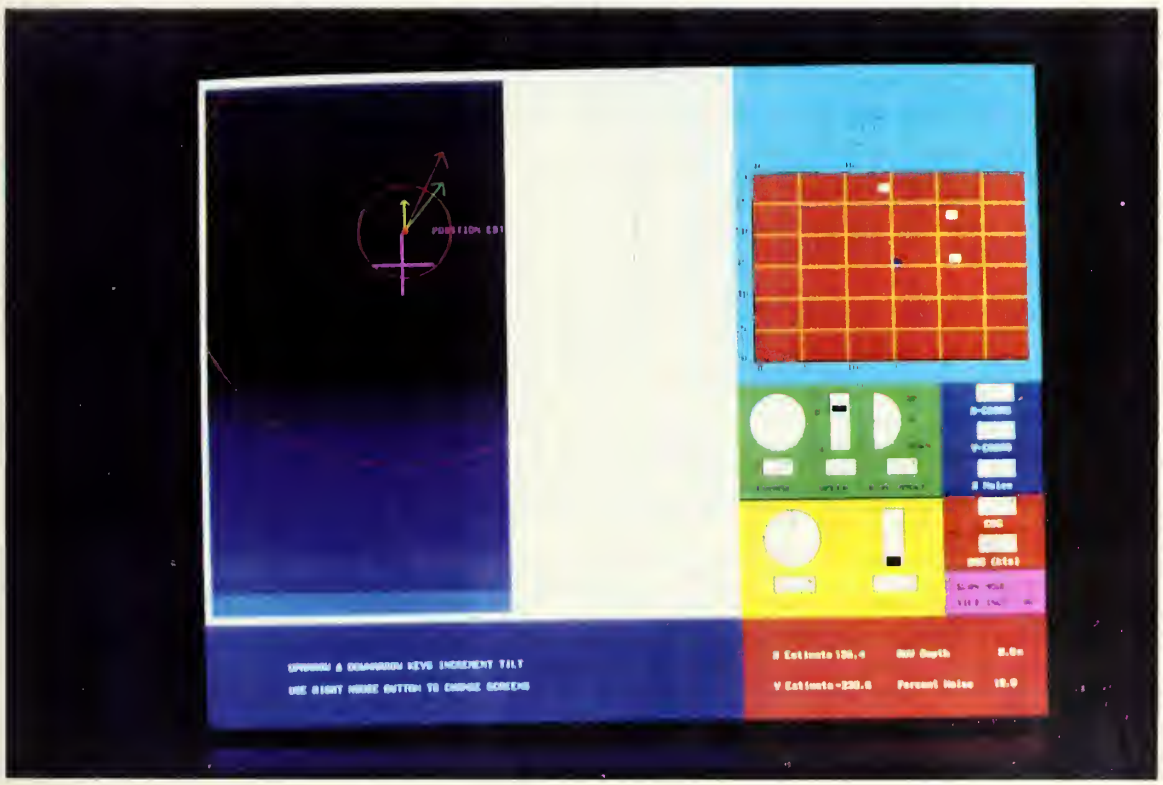

Figure 12. Scan/Position Estimation Screen

## E. SIMULATION RESULTS

The following graphs show the accuracy of the navigator under various environmental conditions. The initial graphs are for the vehicle transiting the pool diagonally from approximately coordinates (0,0) in the shallow end to (202, 354) in the deep end. A set of 000° was applied with drift speeds of 0, 0.3, 0.5, 0.8 or 1.00 knot with a ten percent sonar error. Vehicle speed was 2 knots with the course adjusted depending on the drift speed to provide a course over the ground of 030°.

A diagonal path with a northerly set was chosen because it represents a severe test of the system. This path is difficult because the error is applied in the y direction and the only source of positional information in the y direction for two-thirds of the test is the fathometer. For the last third, more reliable y information is available when sonar return information is available from the deep end wall. A large value of sonar error, 10%, was chosen to provide a reasonable test of the navigator's ability to correct for such noise. Figures 13 through 17 show a comparison between a navigator which has a Kalman filter and a navigator which is using raw observation data for position estimation correction. The filter was designed to compensate more for sonar measurement errors than for set and drift. This can be seen upon observation of the graph with 1 knot of drift. The unfiltered simulation (Figure 17) tracks better during the initial part of the run where only the fathometer is capable of detecting positional errors in the y direction caused by the drift. As the vehicle nears the north wall, sonar returns from the wall reduce the overall positional error but the filtered estimate

becomes more accurate because the primary source of error is the sonar noise. In the cases where the drift is less than 50 percent of vehicle speed (less than 1 knot) the filtered simulation provided a better estimation of vehicle position. Figures 18 and 19 show a comparison between the filtered and unfiltered estimates versus the actual path for a drift of .08 knot. Even though in this case the absolute error of the two estimators are not significantly different, the filtered estimator curve is much smoother and would provide input with less transients to the autopilot which will use this information. Figure 20 shows the case where the vehicle is transiting across the pool with no set and drift, but with 10 percent sonar noise. It can be seen that the filtered system successfully removes the noise but in the unfiltered simulation the noise causes significant positional error. The massive position error in this case was caused by the unfiltered navigator believing it was near the right shallow corner because of the sonar measurement error.

## F. CONCLUSION

In conclusion it appears that in all cases the filtered estimates are preferable to the unfiltered estimates because the volatility of the filtered observations is less than the unfiltered. Also the accuracy of the navigator even in the severest case can be seen to be to be sufficient to allow the vehicle to maneuver in the test pool.

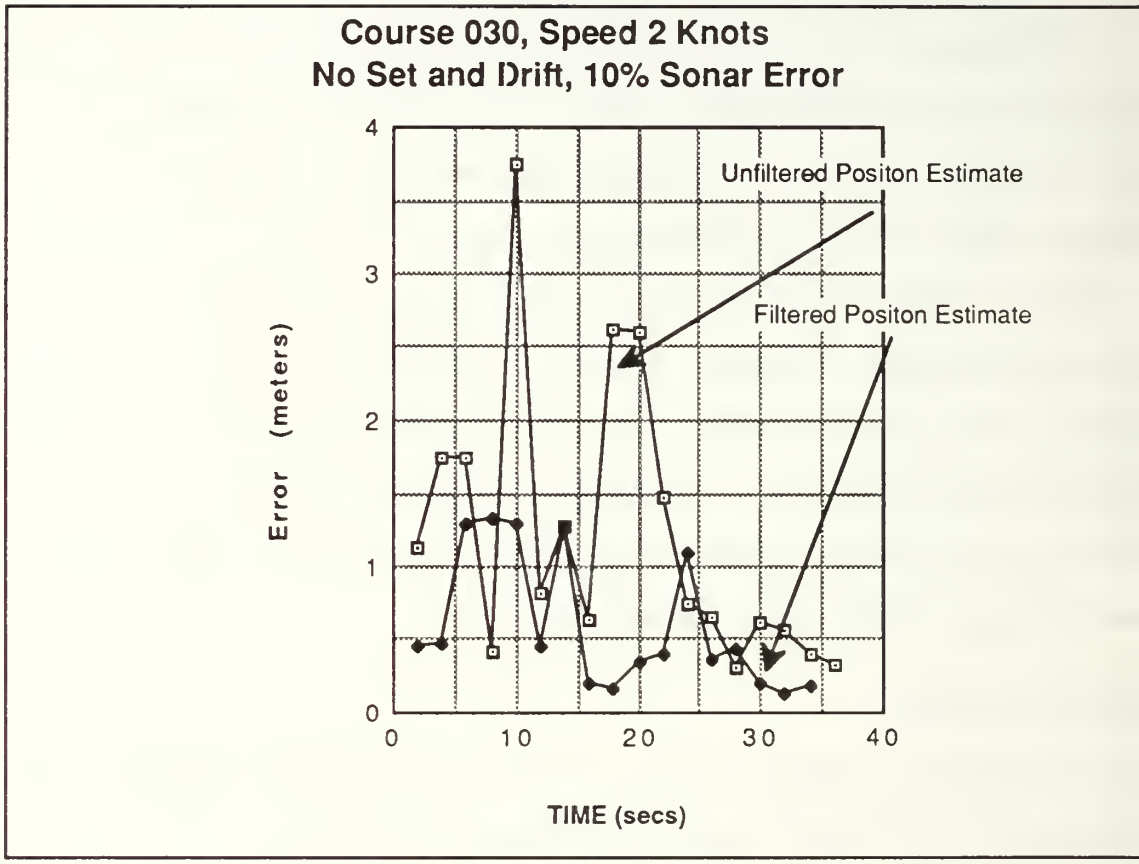


Figure 13. Simulation Run No Set or Drift

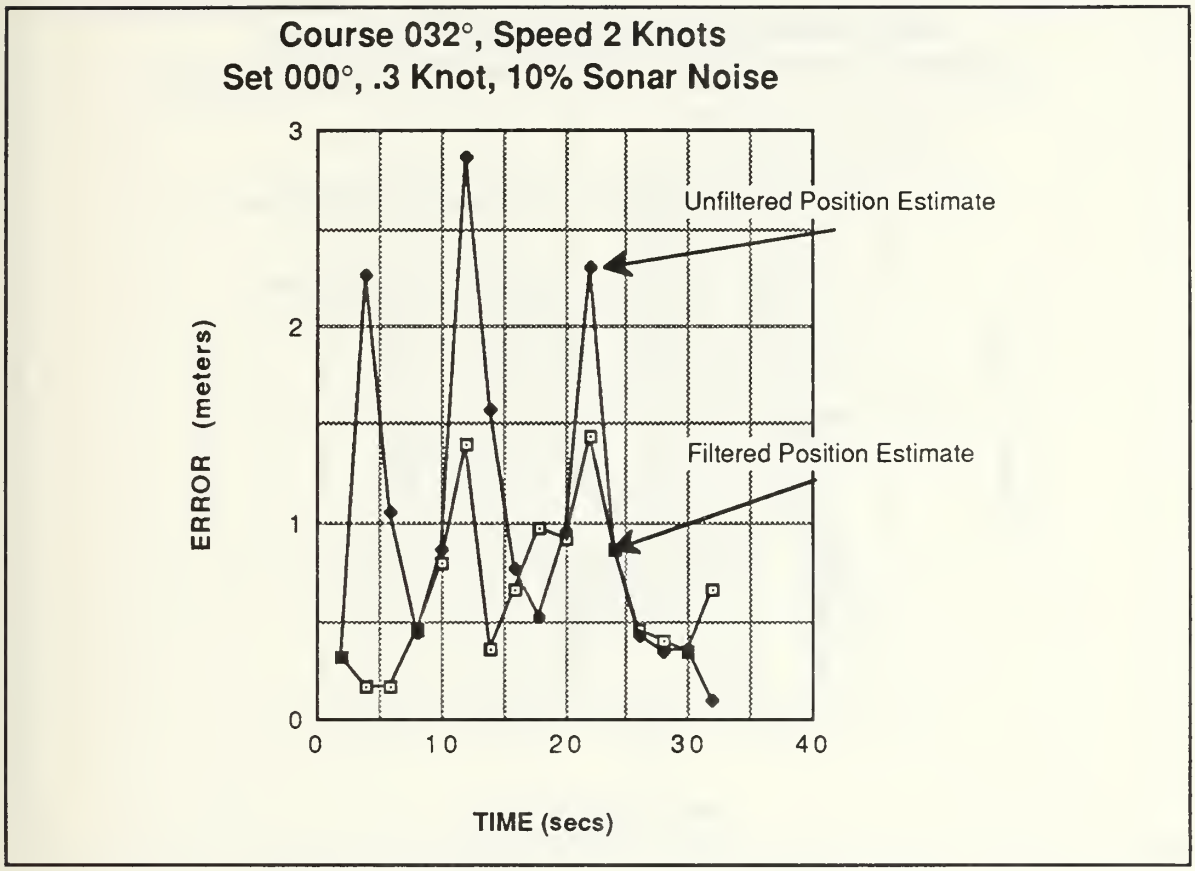


Figure 14. Simulation Run 0.3 Knot Drift

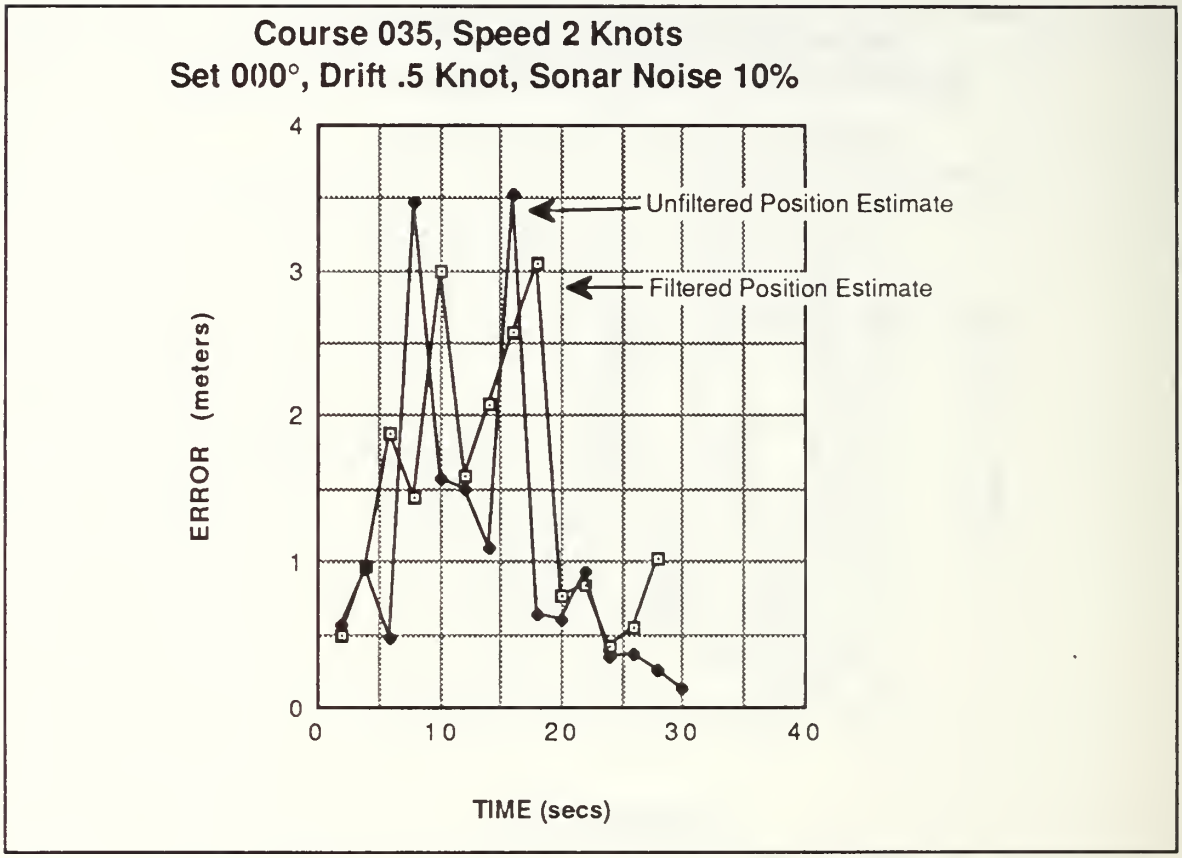


Figure 15. Simulation Run 0.5 Knot Drift

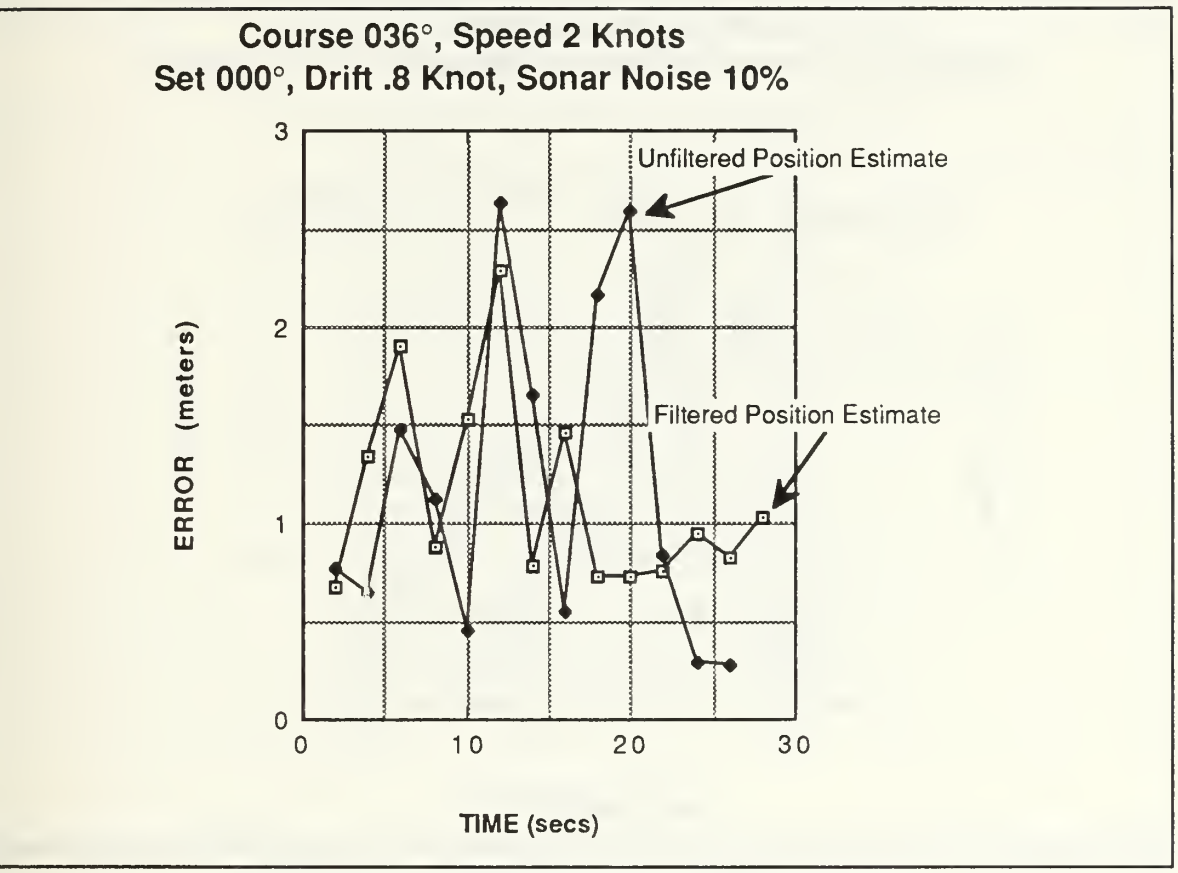


Figure 16. Simulation Run 0.8 Knot Drift

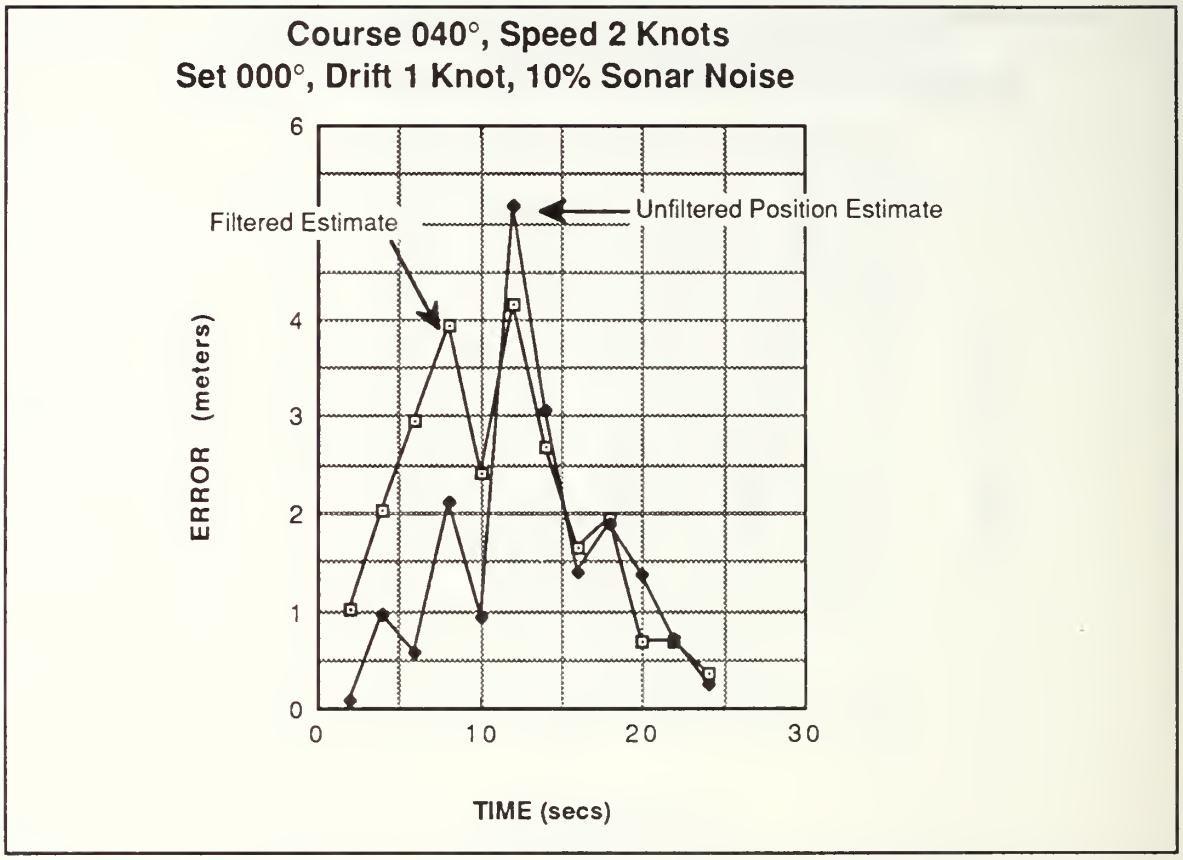


Figure 17. Simulation Run 1.0 Knot Drift

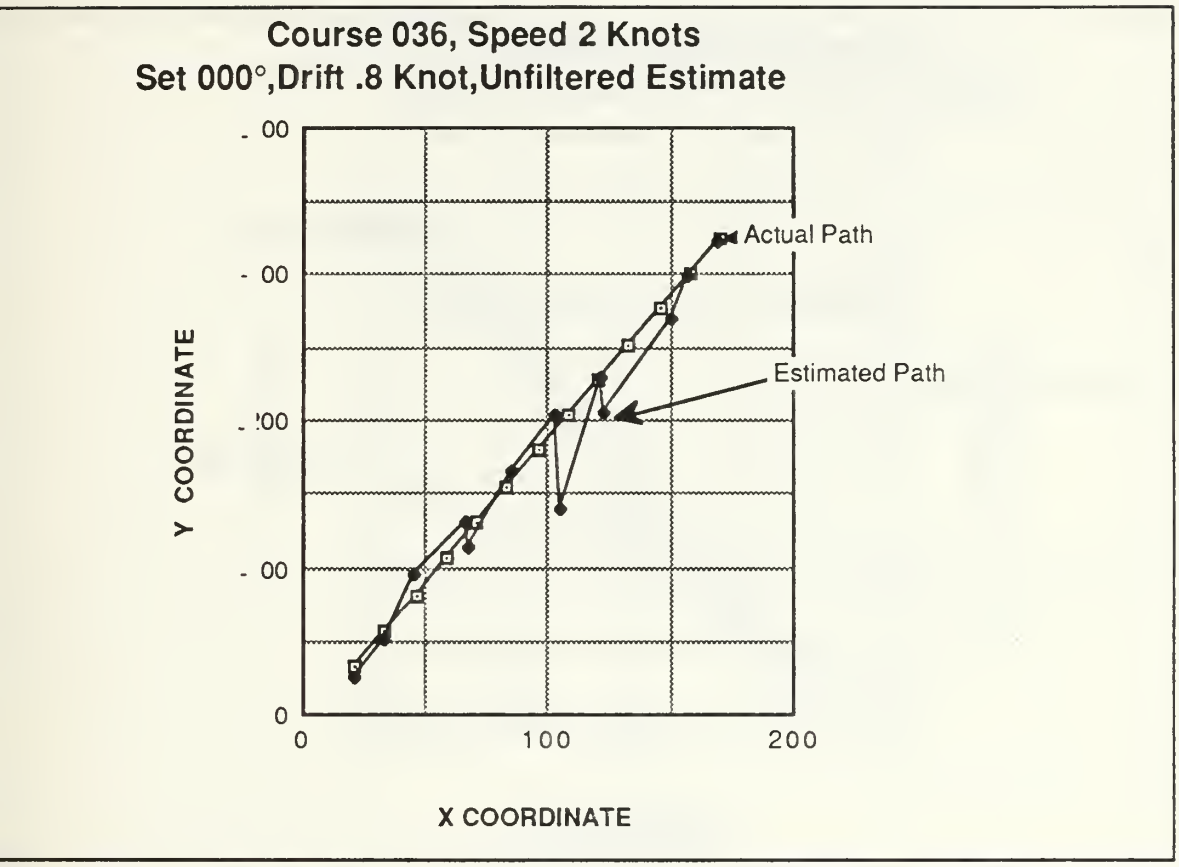


Figure 18. Unfiltered to Actual Comparison

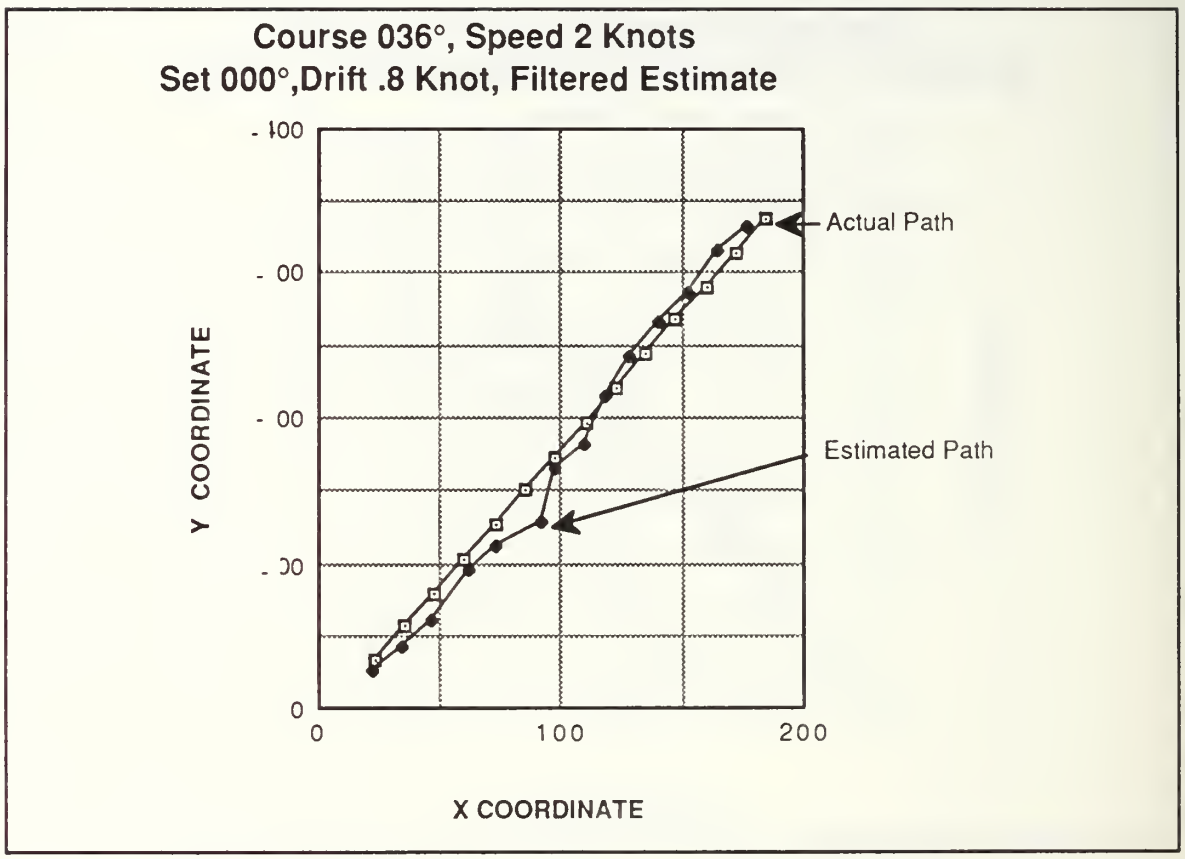


Figure 19. Filtered to Actual Comparison

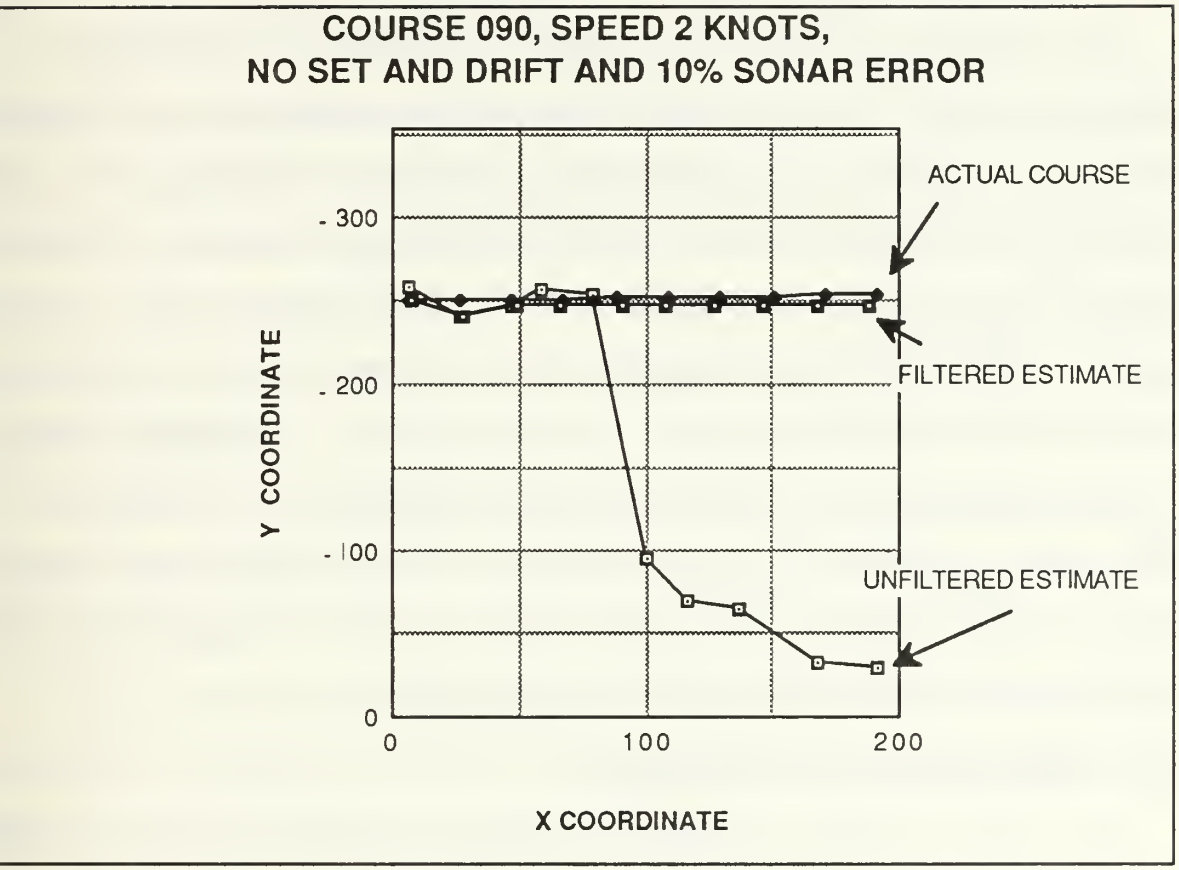


Figure 20. Simulation Run Parallel to x Axis

# V. CONCLUSIONS

## A. ANALYSIS

The simulation results from Chapter 4 demonstrate the navigator's ability to correct for set and drift and sonar measurement errors. Though sonar measurement error contributed to estimate positional errors, in general the primary cause of error was the inability of the sonar to obtain enough terrain information because of the rudimentary design of the sonar. This caused the fathometer to be the only source of terrain information in a significant portion of some of the runs. This design makes it imperative that every attempt be made to minimize sonar noise in the fathometer. A possible way of doing this is to ensure that sonar points directly down to produce the smallest sonar footprint on the bottom and thereby receive the smallest amount of sonar noise.

## B. RESEARCH EXTENSIONS

The most critical research extension of this thesis is the implementation of the navigator program in the actual Testbed AUV. This will involve testing the vehicle sonar to determine what adjustments will be necessary to the covariance of measurement error computation in the filter module and using experience with the actual sonar to make the sonar model in the simulation more accurate. Integration of the navigation program with the AUV's path planner and the control modules will also be necessary.

Inclusion of set and drift measurements as a variable used by the Kalman filter is an area of further study which should be investigated. The drift of the vehicle can be measured using the least squares terrain matching described in this thesis. If the measured error is a consistent drift, it could be included to improve the estimate of the next position.

A research effort of interest would be simulating navigation of the vehicle in an ocean area such as Monterey Bay. In order to obtain a better validation of the bottom-matching estimator, sweeping maneuvers could be conducted to provide a complete bottom scan for comparison with a larger database. Digital terrain maps of the Monterey Bay are available from the NPS IDEA lab and could easily be installed in the present simulator.

Another area of further study would be the use of a neural net for terrain following by the AUV and a comparison of accuracy and speed with this method.

One final extension that should be considered is providing the navigator the ability to exclude sonar sonar returns from its calculations that are known not to be caused by the terrain.

# APPENDIX - PROGRAM LISTING

```
/*+-----------------------------+
|
|       main_sonar.c
|
+-----------------------------+                */
/* main_sonar --  an IRIS-GTX program originally written  by Chet Hartley.  Modified by John Friend to simulate the sonar of the NPS AUV operating in the NPS swimming pool.*/
#include "gl.h"          /* get the graphics defs          */
#include "device.h"      /* get the graphics device defs   */
#include "sonar.h"       /* constants                      */
#include "stdio.h"
/* FHL elevations used as depths      */

int
bottom_data[BOTTOM_POINTS_WIDTH][BOTTOM_POINTS_HEIGHT];

/* arrays to hold successive sonar scan data */

int
right_depth_array[SONAR_RESOLUTION][SONAR_RESOLUTION];

int
left_depth_array[SONAR_RESOLUTION][SONAR_RESOLUTION];
int stop = 1;/*Makes program wait after obs_filter*/
int go = 0;/*Allows program to proceed in free run*/
int delay;
/* max and min depth vars for chart generation */

int max_depth = 0;

int min_depth = 9999;
```

```
short  val;

main()

{

   /* initial submarine location and orientation values */

   float x_coord = 145.0; /*tenths of meters */

   float estx_coord = 145.0; /*tenths of meters */

   /* the y_coord is negative due to orthoganal coord system
   choice */

   float y_coord = -10.0; /* tenths of meters */

   float esty_coord = -10.0; /* tenths of meters */

   /* depth is passed in tenths of meters */

   float auv_depth = 1.0;

   /*float auv_depth = M_PER_FEET * 500.0;  feet converted to
meters */

   int course = 0;

   float speed = 2.0;

   int dive_angle = 0;

   int roll_angle = 0;
   /* variables for changing course, speed and dive_angle */

   int selected_course = 0;

   float selected_speed = 0.0;

   int selected_dive_angle = 0;
```

```
/* initial current values */

int set = 36;

float drift = 1.0;


/* initial cog/sog values */

/* cog = course over ground, sog = speed over ground */

int cog = 0;

float sog = 0.0;
/* boolean for  location change */

short change_location = FALSE;
/* boolean for hoist , down is operational */

short hoist_down= TRUE;
/* boolean for power on */
short power_on = TRUE;
/* initial  sonar configuration */

int sector_setting = 360;

short sector_setting_change=FALSE; /* boolean for sector
change */

int previous_sector_setting= 360;

int range_setting = 150;

short range_setting_change=TRUE; /* bool for range setting
change */
```

```
int previous_range_setting= 800;
int tilt_angle = 0;

int tilt_inc = 90;

short tilt_angle_change = FALSE;

int scan_heading = 0;

short scan_heading_change = FALSE;

int range_ring_location = 0;
short sweep_clockwise = TRUE; /* bool for sweep direction */

int sonar_sweep_location = 225;
int sector_sweep_start = 0;

int sector_sweep_end = 0;
/* the sonar beam length */

float beam_length = 0.0;

float beam_inc = 1.0;

short encountered_contact = FALSE;

short scan_complete = FALSE;

short scan_mode = COMPLETE_SCAN_AND_STORE;


/* boolean for 3 buttons hit on mouse */

short exit = FALSE;
/* boolean for AUV running aground */

short aground_flag = FALSE;
/* boolean for active array toggle */
```

```
/* start with left array active */

short left_depth_array_active = TRUE;
/* vars for holding depth array max/mins */

int right_array_max_depth, right_array_min_depth;

int left_array_max_depth, left_array_min_depth;
/* vars for holding depth array AUV coords */

float left_x_coord, left_y_coord, left_depth;

float right_x_coord, right_y_coord, right_depth;
/* boolean for displaying depth arrays before sweep has
completed */

short request_depth_array_display = FALSE;

FILE *fopen(), *fp;

/* initialize the iris */

init_iris();


mainmenu_ctrls = defpup("SCAN MENU %t|SCAN MODE 0 %1

|SCAN MODE1 %x2|SCAN MODE 2 %x3|SHOW CHART
%x4|SHOW      NEXT

POSITION %x5|STOP AFTER EACH SCAN %x6|FREE
RUNNING %x7

|QUIT %x8");


mainmenu_sets = defpup(" INITIALIZATION MENU %t

|SCAN MODE 0 %x1|SCAN MODE 1 %x2|SCAN MODE 2 %x3
```

```
|0 PERCENT NOISE %x4|1 PERCENT NOISE %x5|5 PERCENT
NOISE      %x6

|10 PERCENT NOISE %x7|COMMENCE SCAN %x8");
/* read the data file into the bottom_data_array */

load_bottom_data();

arg = 1;/*initialize depth_array counter*/

percent_noise = 0.0;   /*initialize noise value*/

g11 = .5;  /*kalman constants*/
g33 = .5;
p11 = .5;
p33 = 1.0;
r11 = 0;
r33 = 0;
/* get initial settings */

get_auv_settings(&scan_mode, &x_coord, &y_coord,
&auv_depth, &course,
&speed, &dive_angle, &selected_course,   &selected_speed,
&selected_dive_angle, &set, &drift, &cog, &sog,
&aground_flag, &tilt_inc);

estx_coord = x_coord;

esty_coord = y_coord;

/* initialize the depth arrays */

initialize_depth_arrays(&sonar_sweep_location,

&left_depth_array_active, &left_array_max_depth,

&left_array_min_depth, &right_array_max_depth,

&right_array_min_depth, x_coord, y_coord, auv_depth,
```

```
&left_x_coord,&left_y_coord, &left_depth, &right_x_coord,

&right_y_coord, &right_depth);


color(BLACK);

                clear();
                make_chart();
                make_estimate_position_plus(estx_coord,esty_coord);


                make_arrows(x_coord,y_coord,auv_depth,course,
                speed,set,drift,cog,sog,sonar_sweep_location,
                beam_length,tilt_angle,range_setting);
make_sonar( range_setting, tilt_angle,

        power_on, x_coord, y_coord,

        auv_depth, course, dive_angle, roll_angle,

        &sonar_sweep_location, sweep_clockwise,

        &beam_length, &beam_inc, &encountered_contact);


/* put up the auv instrument readings */

make_readout(scan_mode, x_coord, y_coord, auv_depth,

course, speed, dive_angle, set, drift, cog, sog, tilt_inc);


make_instructions(aground_flag);
swapbuffers();


while(TRUE)
```

```
{
        if (exit)
                break;
        else
        {
                if (change_location || aground_flag)
                /* get the new location and auv settings */
                {
                        winpop();
                        change_location = FALSE;
                        go = 0;

                        get_auv_settings(&scan_mode,
&x_coord,&y_coord, &auv_depth,
&course, &speed, &dive_angle, &selected_course,
&selected_speed, &selected_dive_angle, &set, &drift, &cog,
&sog, &aground_flag, &tilt_inc);
                                estx_coord = x_coord;
                                esty_coord = y_coord;
                        /* initialize the depth arrays */
```

```
                initialize_depth_arrays(&sonar_sweep_location,

&left_depth_array_active, &left_array_max_depth,

&left_array_min_depth, &right_array_max_depth,

&right_array_min_depth, x_coord, y_coord, auv_depth,

&left_x_coord, &left_y_coord, &left_depth,

&right_x_coord, &right_y_coord, &right_depth);


                make_chart();
                make_estimate_position_plus(estx_coord,esty_coord);

                make_arrows(x_coord,y_coord,auv_depth,course,

                speed,set,drift,cog,sog,sonar_sweep_location,

                beam_length,tilt_angle,range_setting);

                display_position_estimate(&estx_coord,&esty_coord,

&auv_depth, &course);

make_instructions(aground_flag);

                        swapbuffers();


                }

        make_sonar( range_setting, tilt_angle,
```

```
        power_on, x_coord, y_coord,
        auv_depth, course, dive_angle, roll_angle,
        &sonar_sweep_location, sweep_clockwise,
         &beam_length, &beam_inc,
        &encountered_contact);

   /* store the sonar return data in the depth arrays */
   store_return_data(scan_mode, encountered_contact,
    auv_depth,sonar_sweep_location, beam_length, beam_inc,
   tilt_angle,left_depth_array_active, &left_array_max_depth,
   &left_array_min_depth, &right_array_max_depth,
                        &right_array_min_depth,course);

/*sonar sweep control*/
control_sonar_sweep(&sonar_sweep_location,power_on,tilt_angle);

/* check scan mode and determine if a scan has completed */
   bottom_scan_controller(scan_mode, power_on, hoist_down,
   &tilt_angle, &tilt_angle_change, tilt_inc,
   sonar_sweep_location, &scan_complete);
                /* display depth array */
```

```
   display_depth_arrays(scan_mode, drift, tilt_inc,
   tilt_angle, scan_complete, request_depth_array_display,
   range_setting, beam_inc, left_depth_array_active,
   left_array_max_depth, left_array_min_depth,
   right_array_max_depth, right_array_min_depth,
   left_x_coord, left_y_coord, left_depth,
   right_x_coord, right_y_coord, right_depth,auv_depth,
   &estx_coord,&esty_coord);
   /* make observation and correct position estimate */
   observation_and_filter(scan_mode, drift, tilt_inc,
   tilt_angle, scan_complete, request_depth_array_display,
   range_setting, beam_inc, left_depth_array_active,
   left_array_max_depth, left_array_min_depth,
   right_array_max_depth, right_array_min_depth,
   left_x_coord, left_y_coord, left_depth,
   right_x_coord, right_y_coord, right_depth,auv_depth,
   &estx_coord,&esty_coord);

   if(esty_coord >0)
{
   esty_coord = -esty_coord;
}
```

```
            if (scan_complete)
                {
                    /* call chart */
            make_chart();
            make_estimate_position_plus(estx_coord,esty_coord);

            display_position_estimate(&estx_coord,&esty_coord,
&auv_depth, &course);
            make_arrows(x_coord,y_coord,auv_depth,course,
            speed,set,drift,cog,sog,sonar_sweep_location,
            beam_length,tilt_angle,range_setting);
            make_instructions(aground_flag);
                    swapbuffers();
            make_chart();
            make_estimate_position_plus(estx_coord,esty_coord);
            display_position_estimate(&estx_coord,&esty_coord,
&auv_depth, &course);
make_arrows(x_coord,y_coord,auv_depth,course,
speed,set,drift,cog,sog,sonar_sweep_location,
```

```
   beam_length,tilt_angle,range_setting);

   make_instructions(aground_flag);
delay = 0;
while(stop == 1)
{
        read_controls(&scan_mode,
        &tilt_angle_change, &power_on, &hoist_down,
        &tilt_angle, &tilt_inc, &exit,  &selected_course,
        &selected_speed, &selected_dive_angle, &set, &drift,
        &change_location, &stop);


if (course > 360)
        {course = course -360;
        }
if (course < 0)
        {course = course + 360;
        }
                        /* put up the auv instrument readings */
        make_readout(scan_mode, x_coord, y_coord, auv_depth,
        course, speed, dive_angle, selected_course,
        selected_speed, selected_dive_angle, set, drift, cog,
        sog, tilt_inc);
```

```
if(go == 1 && delay > 5)
{
                stop = 0;
}

                                delay = delay +1;
}/*end stop*/

                                stop = 1;

                        } /* end if display arrays */


                /* calculate the next position of the auv */

        get_next_position(scan_mode, &x_coord, &y_coord,

        &auv_depth, &cog, &sog, &course, &speed, &dive_angle,

         selected_course, selected_speed, selected_dive_angle,

         set, drift, &aground_flag, scan_complete);


                /* estimate the next position of the auv */

  get_next_estimate(&estx_coord,

  &esty_coord,&auv_depth, &course,

   &speed,scan_complete,&dive_angle);

                /* check array toggle for update */

                update_depth_arrays(scan_complete, x_coord,

                y_coord, auv_depth, &left_depth_array_active,
```

```
                &left_array_max_depth,
                &left_array_min_depth, &right_array_max_depth,
                &right_array_min_depth, &left_x_coord,
                &left_y_coord, &left_depth, &right_x_coord,
                &right_y_coord, &right_depth);
                if(scan_complete)
{
        /*reset depth_array*/
        reset_depth_array();
}
                        /* reset all bools to false */
                        reset_flags(&scan_complete,
                        &sector_setting_change,
                        &encountered_contact);
                }
        } /* while */
        /* clean up */
```

```
        color(BLACK);
        clear();
        swapbuffers();
        color(BLACK);
        clear();
        swapbuffers();
        finish();


        greset();
        gexit();


}   /* end of main_sonar */
```

```
/*+------------------------
  |
  |  Sonar.h
  |
   +------------------------
*/

#define FEET_PER_M 1.0
#define FAR_CENTER_X 62.0 /* far gain dial center */
#define FAR_CENTER_Y 6.0

#define NEAR_CENTER_X 52.0 /* near gain dial center */

#define NEAR_CENTER_Y 6.0

#define DIAL_RADIUS 2.0 /* radius of black dials on panel */

#define START_TOGGLE_Y 6.0 /* y-coord that all toggles start on */

#define TILT 1 /* tilt toggle */

#define SCAN 2 /* scan toggle */

#define RANGE_RING 3 /* range toggle */

#define UP 1 /* for toggle up/down */

#define DOWN 0

#define SWEEP_STEP 45 /* Three sonar beams 45 degrees apart*/

#define POWER_OFF 3 /* if 3 or less, consider power off */

#define VOLUME_OFF 3 /* if 3 or less, consider volume off */

#define SONAR_RADIUS 44.0  /* world coord sonar radius */

#define SONAR_RADIUS_MINUS_A_LITTLE 43.80 /* radius for
outer
```

```
ring of range ring overlay */

/* next to outer ring */

#define THREE_QTR_SONAR_RADIUS (SONAR_RADIUS*(3.0/4.0))

/* middle ring */

#define HALF_SONAR_RADIUS (SONAR_RADIUS/2.0)

/* inside ring */

#define ONE_QTR_SONAR_RADIUS (SONAR_RADIUS*(1.0/4.0))


#define RESOLUTION_IN_METERS 1.00  /* depth value every 1.00 cm */


#define BOTTOM_POINTS_WIDTH 351

/* bottom data max row */

#define BOTTOM_POINTS_HEIGHT 351


#define RTOD 57.29578  /* radians to degrees conversion factor    */

#define DTOR 0.0174533 /* degrees to radians conversion factor    */


/* define color numbers */

/* for wmask */
```

```
#define ORANGE 9

#define GREY 10

/* for CHART_MASK */

#define A_MAGENTA 1024

#define A_YELLOW 512

#define A_GREEN 256

#define A_RED 128

#define A_BLACK 64


/* cursor constants */

#define CURSOR_COLOR RED

#define MY_CURSOR 1


/* x and y values for all video read out number locations */

#define RANGE_SETTING_ONES_X -27.0

#define RANGE_SETTING_TENS_X -32.0

#define RANGE_SETTING_HUN_X -37.0

#define RANGE_SETTING_Y 44.0


#define RANGE_RING_ONES_X 39.0
```

```
#define RANGE_RING_TENS_X 34.0
#define RANGE_RING_HUN_X 29.0
#define RANGE_RING_Y 44.0

#define TILT_ONES_X -33.0
#define TILT_TENS_X -38.0
#define TILT_Y 38.0

#define DEPTH_ONES_X 39.0
#define DEPTH_TENS_X 34.0
#define DEPTH_HUN_X 29.0
#define DEPTH_Y 38.0
/* number of arcs in one beam */
#define NUMBER_OF_ARCS 15
#define NUMBER_OF_ARCS_MINUS1 (NUMBER_OF_ARCS-1)

/* sonar video colors */
#define NO_CONTACT_COLOR BLUE
#define CONTACT_COLOR RED
#define SHADOW_COLOR CYAN
```

```
#define BOUNDARY_COLOR GREY

/* amount the beam tip will be increased/decreased each time
   depending on the range setting */
#define BEAM_INC_800 (800/NUMBER_OF_ARCS) /* 60.0 */
#define BEAM_INC_600 (600/NUMBER_OF_ARCS) /* 30.0 */
#define BEAM_INC_400 (400/NUMBER_OF_ARCS) /* 26.6 */
#define BEAM_INC_300 (300/NUMBER_OF_ARCS) /* 20.0 */
#define BEAM_INC_200 (200/NUMBER_OF_ARCS) /* 13.3 */
#define BEAM_INC_150 (150/NUMBER_OF_ARCS) /* 10.0 */
#define BEAM_INC_100 (100/NUMBER_OF_ARCS) /* 6.6 */
#define BEAM_INC_60 (60/NUMBER_OF_ARCS) /* 4.0 */
#define BEAM_INC_30 (30/NUMBER_OF_ARCS) /* 2.0 */
#define BEAM_INC_15 (15/NUMBER_OF_ARCS) /* 1.0 */

#define PI 3.1416
#define HALF_PI 1.5708

/* returned when beam tip reaches database boundry */
#define BOUNDARY_FLAG_VALUE 999999
```

```
/* max and min auv speeds */

#define MAX_SPEED 6  /* 6 kts */

#define MIN_SPEED -4 /* 4 kts astern */


/* world to screen speed increments on speed bar */

/*      SPEED_INC      (speed_ortho_bar_range(71-
53)/(MAX_SPEED_AHEAD - MIN_SPEED)) */

#define SPEED_INC (18/(MAX_SPEED - MIN_SPEED))


/* location in y of the zero mark on the speed bar */

#define ZERO_Y (71.0 - (SPEED_INC * MAX_SPEED))


/* current bounds */

#define MAX_DRIFT 6 /* 6 knots */

#define DRIFT_INC (18/MAX_DRIFT) /* 3 */


/* constants for depth to color conversion */

#define MAX_COLOR_NUMBER 63

#define MAX_DEPTH 2200

#define NUMBER_OF_COLORS 47
```

```
/* constants for creating the direction/velocity arrows */
#define ARROW_FACTOR 20
#define ARROW_WING_FACTOR (ARROW_FACTOR * 0.20)

/* constant determining size of the position plus on chart */
#define PLUS_FACTOR 4

/* writemask number for objects over the chart */

/* knots to meters conversion */
#define KTS_TO_METERS ((1.0/3600.0)*18560.54) /* hr/sec * m/kt
*/

/* amount of time between buffer swaps */
#define SMALL_TIME_INC 0.50  /* half of a second */
#define LARGE_TIME_INC 2.0  /* twenty seconds */

/* gains for speed, course, and dive angle dynamic changes */
```

```
    #define SPEED_CHANGE_GAIN 0.25  /* 0.25 knots per TIME_INC
*/

    #define COURSE_CHANGE_GAIN 20       /* 20 degrees per
TIME_INC */

    #define DIVE_ANGLE_CHANGE_GAIN 1 /* 1 degree per
TIME_INC */


    /* 300 by 300 array to hold sonar return data */

    #define SONAR_RESOLUTION 300


    /* constants for the depth array display */

    #define Y_0 5.5

    #define Y_5 12.5

    #define Y_10 20.5

    #define Y_15 28.0

    #define Y_20 35.5

    #define Y_25 42.5

    #define Y_29 48.5


    #define X_N_0 3.5

    #define X_N_5 11.0

    #define X_N_10 18.0
```

```
#define X_N_15 25.5

#define X_N_20 33.0

#define X_N_25 40.5

#define X_N_29 46.0


#define X_O_0 52.5

#define X_O_5 60.0

#define X_O_10 67.0

#define X_O_15 74.5

#define X_O_20 82.0

#define X_O_25 89.5

#define X_O_29 95.0


#define Y_BOTTOM 3.0

#define Y_TOP 51.0


#define BASE_NEW_X 5.0

#define BASE_NEW_Y 3.0

#define BASE_OLD_X BASE_NEW_X

#define BASE_OLD_Y 52.0
```

```
/* The 3 different scan modes */

#define JUST_SCAN_NO_STORE 0

#define ONE_SCAN_AND_STORE 1

#define COMPLETE_SCAN_AND_STORE 2


#define MAX_PASSES 6


#define MAX_INTEGER 9999999

#define CRIT_SIZER 100

#define CRIT_SIZEC 100

#define MAXANGLE .78

int mainmenu_ctrls,mainmenu_sets,mainmenu_arrays;

int arg,go;

int depth_array[7][3];
int win_id_wesmar,win_id_array,win_id_readout;


float
g11,g33,p11,p33,r11,r33,q11,q33,x_dist1,x_dist2,x_dist3,y_dist1,y_dist2,y_di
st3,percent_noise;
float criterion_array[CRIT_SIZER][CRIT_SIZEC];

float au,ar,ak1,ak2,ax1,ax2,ac1,ac2,aphi11,aphi12,aphi21,aphi22,
ax1plus,ax2plus,adel1,adel2,ay;
```

```
/*          +-----------------------------+
            |
            |       disp_arrays.c
            |
            +-----------------------------+                  */


/* Displays the depth arrays */
#include "sonar.h"
#include "gl.h"
#include "math.h"
#include "device.h"


display_depth_arrays(scan_mode, drift, tilt_inc,
   tilt_angle, scan_complete, request_depth_array_display,
   range_setting, beam_inc, left_depth_array_active,
   left_array_max_depth, left_array_min_depth,
   right_array_max_depth,
   right_array_min_depth, left_x_coord, left_y_coord,        left_depth,
   right_x_coord, right_y_coord, right_depth, auv_depth,
   estx_coord, esty_coord)


   short scan_complete, request_depth_array_display,
   left_depth_array_active, scan_mode;

   int range_setting, tilt_inc, tilt_angle;

   int left_array_max_depth, left_array_min_depth,
         right_array_max_depth, right_array_min_depth;

   float beam_inc, drift;

   float left_x_coord, left_y_coord, left_depth,

         right_x_coord, right_y_coord, right_depth, auv_depth,
```

```
        *estx_coord,*esty_coord;


    {

    extern int
bottom_data[BOTTOM_POINTS_HEIGHT][BOTTOM_POINTS_WIDT
H];

    extern int
left_depth_array[SONAR_RESOLUTION][SONAR_RESOLUTION];

    extern int
right_depth_array[SONAR_RESOLUTION][SONAR_RESOLUTION];

    extern int max_depth, min_depth;

       char str[40];

       int
i,
j,row,col,r,c,row_start,col_start,bottom_value,k,rval,min_rval,cval,min_cval
;

       int array_depth, e_cog, a_cog, water_depth_at_ave;

       float cog_x, cog_y, a_distance, e_distance, ave_x, ave_y,

            depth_below_auv,zestx_coord,zesty_coord,num;

       Colorindex colour;

       short first_pass, too_many_passes, new_min_criterion,

            analysis_completed, first_match;

    int number_of_cells, number_of_passes, row_offset, col_offset,

            cell_location, row_inc, col_inc,row_change,col_change;
```

```
    float min_criterion, x_dist, y_dist;

    short right_array_empty, left_array_empty;


   int hititem;

    short value;

    if (scan_complete)

    /* display the depth arrays */

    {
        first_pass = TRUE;
  winset((long) win_id_array);
  winpop();

  /*viewport(632*XMAXSCREEN/1000,XMAXSCREEN,511*YMAX
SCREEN/1000,YMAXSCREEN);
  */

        ortho2(0.0,50.0,0.0,75.0);

    /*

        color(BLACK);

        clear();

        swapbuffers();
  */


                color(CYAN);

                clear();

                /* title info first */
```

```
color(BLACK);
cmov2(19.0,70.0);
charstr("SONAR RANGE ARRAY");

cmov2(19.0, 64.0);
charstr("X_COORD:");
cmov2(19.0, 62.0);
charstr("Y_COORD:");
cmov2(19.0, 60.0);
charstr("DEPTH:");



cmov2(19.0, 56.0);
charstr("COG:");


/* build the array boxes */
/* first, the numbers along the sides */
/* new scan, left side */
cmov2(1.0, Y_0);
```

```
charstr("0");
cmov2(1.0, Y_5);
charstr("5");
cmov2(0.5, Y_10);
charstr("10");
cmov2(0.5, Y_15);
charstr("15");
cmov2(0.5, Y_20);
charstr("20");
cmov2(0.5, Y_25);
charstr("25");
cmov2(0.5, Y_29);
charstr("29");
/* new scan bottom */
cmov2(X_N_0, Y_BOTTOM);
charstr("0");
cmov2(X_N_5, Y_BOTTOM);
charstr("5");
cmov2(X_N_10, Y_BOTTOM);
charstr("10");
```

```
cmov2(X_N_15, Y_BOTTOM);
charstr("15");
cmov2(X_N_20, Y_BOTTOM);
charstr("20");
cmov2(X_N_25, Y_BOTTOM);
charstr("25");
cmov2(X_N_29, Y_BOTTOM);
charstr("29");

/* new scan top */
cmov2(X_N_0, Y_TOP);
charstr("0");
cmov2(X_N_5, Y_TOP);
charstr("5");
cmov2(X_N_10, Y_TOP);
charstr("10");
cmov2(X_N_15, Y_TOP);
charstr("15");
cmov2(X_N_20, Y_TOP);
charstr("20");
```

```
        cmov2(X_N_25, Y_TOP);
        charstr("25");
        cmov2(X_N_29, Y_TOP);
        charstr("29");


        /* now display array*/
        /* first determine active array.
Its contents will be displayed on the left (NEW)
and the inactive array on the right (OLD) */

        /* fill backg ounds */
        color(RED);
        /* new */
        rectf(3.0,5.0,48.0,50.0);
        /* old */
        rectf(52.0,5.0,97.0,50.0);

        if (left_depth_array_active)
        { /* left array has new scan info */
                for (i=0; i<SONAR_RESOLUTION; ++i)
```

```
                {
                    for (j=0; j<SONAR_RESOLUTION; ++j)
                        {
                /* first fill in new array */
                array_depth = left_depth_array[i][j];
                if (array_depth != 99999)
                { /* we have contact depth data */
                /* fill in that rectangle */
                colour = (Colorindex)(MAX_COLOR_NUMBER -
                ((((float)array_depth - min_depth)/
                (max_depth - min_depth)) *
NUMBER_OF_COLORS));
                color(colour);
                rectf(BASE_NEW_Y + (.1*j * 1.5),
                BASE_NEW_X + (.1*i * 1.5),
                BASE_NEW_Y + ((.1*j + 1) * 1.5),
                BASE_NEW_X + ((.1*i + 1) * 1.5));
                        }
                    } /* end for */
                } /* end for */
```

```
                /* put up AUV position */
                color(BLACK);
                /* display new coord info */
                sprintf(str, "%4.1f", left_x_coord);
                cmov2(27.0,64.0);
                charstr(str);
                sprintf(str, "%4.1f", -left_y_coord);
                cmov2(27.0,62.0);
                charstr(str);
                sprintf(str, "%4.1f", left_depth);
                cmov2(27.0,60.0);
                charstr(str);

/* calc actual cog and distance traveled */
cog_x = left_x_coord - right_x_coord;
cog_y = left_y_coord - right_y_coord;
a_distance = sqrt((cog_x * cog_x)+(cog_y *
cog_y));
                /* cog */
                /* 9 cases */
                if ((cog_x == 0) && (cog_y == 0))
```

```
        a_cog= 0;

else if ((cog_x == 0) && (cog_y > 0))

        a_cog= 0;

else if ((cog_x == 0) && (cog_y < 0))

        a_cog= 180;

else if ((cog_y == 0) && (cog_x < 0))

        a_cog= 270;

else if ((cog_y == 0) && (cog_x > 0))

        a_cog= 90;

else if ((cog_x < 0) && (cog_y > 0)) /*
270 - 360 */

        a_cog= 270 + (int)(atan(cog_y/(-1
        * cog_x)) * RTOD);

else if ((cog_x < 0) && (cog_y < 0)) /*
180 - 270 */

a_cog= 270 - (int)(atan(cog_y/cog_x) *
RTOD);

else if ((cog_x > 0) && (cog_y > 0)) /* 0
- 90 */

a_cog= 90 - (int)(atan(cog_y/cog_x) *
RTOD);

else if ((cog_x > 0) && (cog_y < 0)) /*
90 - 180 */
```

```
        a_cog= 90 + (int)(atan((-1 *
cog_y)/cog_x) * RTOD);


        sprintf(str, "%3d", a_cog);

        cmov2(27.0,56.0);

        charstr(str);


} /* end if left active */

else /* right array active */

{ /* right array has new scan info */

        for (i=0; i<SONAR_RESOLUTION; ++i)

        {

        for (j=0; j<SONAR_RESOLUTION; ++j)

                {

        /* first fill in new array */

        array_depth = right_depth_array[i][j];

        if (array_depth != 99999)

        { /* we have contact depth data */

        /* fill in that rectangle */

        colour =
        (Colorindex)(MAX_COLOR_NUMBER -
```

```
        ((((float)array_depth - min_depth)/
        (max_depth - min_depth))
        *NUMBER_OF_COLORS));
        color(colour);
rectf(BASE_NEW_Y + (.1*j * 1.5),
BASE_NEW_X + (.1*i * 1.5),
BASE_NEW_Y + ((.1*j + 1) * 1.5),
BASE_NEW_X + ((.1*i + 1) * 1.5));
                        }

                } /* end for */
        } /* end for */

        /* put up AUV position  */
        color(BLACK);


        /* display new coord info */
        sprintf(str, "%4.1f", right_x_coord);
        cmov2(27.0,64.0);
        charstr(str);
```

```
sprintf(str, "%4.1f",-right_y_coord);

cmov2(27.0,62.0);

charstr(str);

sprintf(str, "%4.1f", right_depth);

cmov2(27.0,60.0);

charstr(str);


/* calc cog and distance traveled */

cog_x = right_x_coord - left_x_coord;

cog_y = right_y_coord - left_y_coord;

a_distance = sqrt((cog_x *
cog_x)+(cog_y * cog_y));
/* cog */

/* 9 cases */

if ((cog_x == 0) && (cog_y == 0))

        a_cog = 0;

else if ((cog_x == 0) && (cog_y > 0))

        a_cog = 0;

else if ((cog_x == 0) && (cog_y < 0))

        a_cog = 180;

else if ((cog_y == 0) && (cog_x < 0))
```

```
                    a_cog = 270;

                else if ((cog_y == 0) && (cog_x > 0))

                    a_cog = 90;

                else if ((cog_x < 0) && (cog_y > 0)) /*
                270 - 360 */

                a_cog = 270 + (int)(atan(cog_y/(-
                1 * cog_x)) * RTOD);

                else if ((cog_x < 0) && (cog_y < 0)) /*
            180 - 270 */

            a_cog = 270 -
            (int)(atan(cog_y/cog_x) * RTOD);
            else if ((cog_x > 0) && (cog_y > 0)) /* 0 - 90
            */

    a_cog = 90 - (int)(atan(cog_y/cog_x) * RTOD);

    else if ((cog_x > 0) && (cog_y < 0)) /* 90 - 180 */

    a_cog = 90 + (int)(atan((-1 * cog_y)/cog_x) *
RTOD);



                sprintf(str, "%3d", a_cog);

                cmov2(27.0,56.0);

                charstr(str);



            } /* end if right active */
```

```
                    /* now overlay grid & boarders */
                    color(YELLOW);
                    linewidth(1);
                    for (i=0; i<5; ++i)
                    {
                        /* new scan verticles */
                        move2(10.5 + (i * 7.5), 5.0);
                        draw2(10.5 + (i * 7.5), 50.0);

                        /* new scan horizontals */
                        move2(3.0, 12.5 + (i * 7.5));
                        draw2(48.0, 12.5 + (i * 7.5));

                    }
                    /* boarders */
                    linewidth(2);
                    color(BLACK);
                    rect(3.0,5.0,48.0,50.0);

                    swapbuffers();
}
/*end scan complete*/
```

```
winset((long) win_id_wesmar);

}/*end display_depth_arrays*/
```

```
/*+-----------------
|
|
|  disp_estimate.c
|  |
|
|
 +-----------------      */

/*Displays an estimate of vehicle positon*/

#include "sonar.h"

#include "gl.h"

#include "math.h"


display_position_estimate(estx_coord,esty_coord,auv_depth,course)

float *estx_coord,*esty_coord,*auv_depth;

int *course;

{
        char str[10];

viewport(632*XMAXSCREEN/1000,XMAXSCREEN,0,156*YMAX
SCREEN/1000);

        ortho2(0.0,100.0,0.0,33.0);

        color(RED);

        clear();

        color(WHITE);

        cmov2(10,20);
```

```
        charstr("X Estimate");


        sprintf(str,"%5.1f",*estx_coord);
        cmov2(30,20);

        charstr(str);

        cmov2(10,10);

        charstr("Y Estimate");


        sprintf(str,"%5.1f",*esty_coord);
        cmov2(30,10);

        charstr(str);

        cmov2(50,20);

        charstr("AUV Depth");


        sprintf(str,"%5.1f",*auv_depth/10);
        cmov2(80,20);

        charstr(str);

        cmov2(90,20);

        charstr("m");

        cmov2(50,10);

        charstr("Percent Noise");


    sprintf(str,"%5.1f",percent_noise*100);
```

```
            cmov2(80,10);
            charstr(str);
    } /*End display position estimate*/
```

```
/*+-----------------
|
|
|  disp_estimate.c
|   |
|
|
 +-----------------      */

/*Displays an estimate of vehicle positon*/

#include "sonar.h"

#include "gl.h"

#include "math.h"


display_position_estimate(estx_coord,esty_coord,auv_depth,course)

float *estx_coord,*esty_coord,*auv_depth;

int *course;

{
        char str[10];

viewport(632*XMAXSCREEN/1000,XMAXSCREEN,0,156*YMAX
SCREEN/1000);

        ortho2(0.0,100.0,0.0,33.0);

        color(RED);

        clear();

        color(WHITE);

        cmov2(10,20);
```

```
        charstr("X Estimate");


        sprintf(str,"%5.1f",*estx_coord);
        cmov2(30,20);

        charstr(str);

        cmov2(10,10);

        charstr("Y Estimate");


        sprintf(str,"%5.1f",*esty_coord);
        cmov2(30,10);

        charstr(str);

        cmov2(50,20);

        charstr("AUV Depth");


        sprintf(str,"%5.1f",*auv_depth/10);
        cmov2(80,20);

        charstr(str);

        cmov2(90,20);

        charstr("m");

        cmov2(50,10);

        charstr("Percent Noise");


    sprintf(str,"%5.1f",percent_noise*100);
```

```
        cmov2(80,10);
        charstr(str);
} /*End display position estimate*/
```

```
/*          +-----------------------------+
            |
            |       get_posit.c
            |
            +-----------------------------+                */

/* This procedure is called by main_sonar to get the next
   position of the auv.  This procedure contains the auv dynamics. */

#include "gl.h"                 /* graphics lib defs */
#include "sonar.h"                  /* sonar constants */
#include "math.h"               /* math definitions */

get_next_position(scan_mode, x_coord, y_coord, depth, cog, sog,
   course, speed, dive_angle, selected_course, selected_speed,
   selected_dive_angle, set, drift, aground_flag, scan_complete)

   int *course, selected_course, *dive_angle, selected_dive_angle,
         *cog, set;
   float *x_coord, *y_coord, *depth, *speed, selected_speed,
         *sog, drift;
   short scan_mode, *aground_flag, scan_complete;

{
   extern int
bottom_data[BOTTOM_POINTS_WIDTH][BOTTOM_POINTS_HEIGH
T];
   static short auv_change = TRUE;/* boolean for auv or current
change */
   static int last_set = 0;
   static float last_drift = 0.0;
   /* vars to calculate cog and sog */
   float course_x, course_y, current_x, current_y,
         cog_x, cog_y;

   if ( scan_complete)
   { /* a sonar scan has completed, update config and position */

         if (selected_course != *course) /* course needs changing */
         {
```

```
                auv_change = TRUE;
                /* determine which way is better to turn to reach
selected
                        course fastest */
                if (selected_course > 180)
                {
                        if ((selected_course > *course) && (*course >=
(selected_course - 180)))
                        {
                                *course = *course +
COURSE_CHANGE_GAIN;

                        }
                        else
                        {
                                *course = *course -
COURSE_CHANGE_GAIN;
                        }
                }
                else
                {
                        if ((selected_course < *course) && (*course <=
(selected_course + 180)))
                        {
                                *course = *course -
COURSE_CHANGE_GAIN;
                        }
                        else
                        {
                                *course = *course +
COURSE_CHANGE_GAIN;
                        }
                }
                if (*course == 360)
                        *course = 0;
                if (*course == -1)
                        *course = 359;

                if(*course - selected_course <
COURSE_CHANGE_GAIN)
```

```
                    *course = selected_course;
                if(selected_course - *course <
COURSE_CHANGE_GAIN)
                    *course = selected_course;

            }

            if (selected_speed != *speed) /* speed needs changing */
            {
                auv_change = TRUE;
                if (selected_speed > *speed)
                    *speed = *speed + SPEED_CHANGE_GAIN;
                else
                    *speed = *speed - SPEED_CHANGE_GAIN;
            }
            if ((*speed > (selected_speed - SPEED_CHANGE_GAIN))
&&
                (*speed < (selected_speed +
SPEED_CHANGE_GAIN)))
                *speed = selected_speed;

            if (selected_dive_angle != *dive_angle) /* dive needs
changing */
            {
                auv_change = TRUE;
                if (selected_dive_angle > *dive_angle)
                    *dive_angle = *dive_angle +
DIVE_ANGLE_CHANGE_GAIN;
                else
                    *dive_angle = *dive_angle -
DIVE_ANGLE_CHANGE_GAIN;
            }

            if (last_set != set)
            {
                auv_change = TRUE;
                last_set = set;
            }

            if (last_drift != drift)
```

```
{
        auv_change = TRUE;
        last_drift = drift;
}

if (auv_change)  /* calc new cog/sog */
{
        /* calculate cog/sog */
        course_x = sin(*course * DTOR) * *speed;
        course_y = cos(*course * DTOR) * *speed;
        current_x = sin(set * DTOR) * drift;
        current_y = cos(set * DTOR) * drift;
        cog_x = course_x + current_x;

        cog_y = course_y + current_y;


        /* 9 cases */
        if ((cog_x == 0) && (cog_y == 0))
                *cog = *course;
        else if ((cog_x == 0) && (cog_y > 0))
                *cog = 0;
        else if ((cog_x == 0) && (cog_y < 0))
                *cog = 180;
        else if ((cog_y == 0) && (cog_x < 0))
                *cog = 270;
        else if ((cog_y == 0) && (cog_x > 0))
                *cog = 90;
        else if ((cog_x < 0) && (cog_y > 0)) /* 270 - 360 */
                *cog = 270 + (int)(atan(cog_y/(-1 * cog_x)) *
                RTOD);
        else if ((cog_x < 0) && (cog_y < 0)) /* 180 - 270 */
                *cog = 270 - (int)(atan(cog_y/cog_x) *
        RTOD);
        else if ((cog_x > 0) && (cog_y > 0)) /* 0 - 90 */
                *cog = 90 - (int)(atan(cog_y/cog_x) * RTOD);
        else if ((cog_x > 0) && (cog_y < 0)) /* 90 - 180 */
                *cog = 90 + (int)(atan((-1 * cog_y)/cog_x) *
                RTOD);
```

```
                /* calculate sog */
                *sog = sqrt((cog_x * cog_x) + (cog_y * cog_y));

        } /* if auv_change */


        /* calculate the new x_coord, y_coord, and depth */
                *x_coord = *x_coord + (*sog * KTS_TO_METERS *
                LARGE_TIME_INC *
                        cos(*dive_angle*DTOR) * cos((*cog - 90)*
                        DTOR));

                *y_coord = *y_coord + (*sog * KTS_TO_METERS *
                LARGE_TIME_INC * cos(*dive_angle*DTOR) *
                        sin((*cog - 90)* DTOR));

                *depth = *depth + (*sog * KTS_TO_METERS *
                        LARGE_TIME_INC *sin(*dive_angle*DTOR));

        /* assume AUV not aground and in operating area */
        *aground_flag = FALSE;

        /* check to see if AUV is in operating area */
        if ((*x_coord < 0.0) || (*x_coord > 1000.0) ||
                (*y_coord > 0.0) || (*y_coord < -1000.0))
                /* y_coord is a negative number */
                *aground_flag = TRUE;

        /* check to see if AUV is aground */
        if ((FEET_PER_M * *depth) >=
                bottom_data[(int)((-1 *
  *y_coord)/RESOLUTION_IN_METERS)]
        [(int)(*x_coord/RESOLUTION_IN_METERS)])
        *aground_flag = TRUE;

   } /* end if scan_complete or JUST_SCAN mode */

   auv_change = FALSE; /* reset, assume no change next time */

} /* end get posit */
```

```
float achange(aCOURSE_CHANGE)
int aCOURSE_CHANGE;
{
                int j;
                for(j=0;j<40;++j)
{
                au = (float) ar * ak1 - (ak1*ax1+ak2*ax2);

                if (au > MAXANGLE)
{
                au = MAXANGLE;
}
                if (au < -MAXANGLE)
{
                au    = -MAXANGLE;
}
     ax1plus = aphi11 * ax1 + aphi12*ax2 + adel1 * au;
                ax2plus = aphi21 * ax1 + aphi22*ax2 + adel2 * au;
                ay = ac1* ax1plus + ac2 * ax2plus;
                ax1 = ax1plus;
                ax2 = ax2plus;
                aCOURSE_CHANGE = ay;
}
}
```

```
/*          +----------------------------+
            |
            |       get_sets.c
            |
            +----------------------------+                */

/* This procedure is called by main_sonar to get the
   values from the operator's controls (mouse and dials)when a
   change in location/depth is requested */

#include "gl.h"                 /* graphics lib defs */
#include "sonar.h"                  /* sonar constants */

get_auv_settings(scan_mode, x_coord, y_coord, depth, course, speed,
   dive_angle, selected_course, selected_speed, selected_dive_angle,
   set, drift, cog, sog, aground_flag, tilt_inc)

   int *course, *dive_angle, *set, *cog, *selected_course,
         *selected_dive_angle, *tilt_inc;
   float *x_coord, *y_coord, *speed, *depth,
         *drift, *sog, *selected_speed;
   short *scan_mode, *aground_flag;

{
   /*initialize noise generator*/

float gasdev();
float junk;
 int idum = -1;
   short done=FALSE;
junk = gasdev(&idum);



   /* build screen */
   /* make auv instrument readout */
   make_readout(*scan_mode, *x_coord, *y_coord, *depth, *course,
*speed,
         *dive_angle, *selected_course, *selected_speed,
         *selected_dive_angle, *set, *drift, *cog, *sog, *tilt_inc);
```

```
        /* make the depth indicator bar */
        make_depth_bar(*x_coord, *y_coord, *depth);
         /* make the instruction billboard */
        make_instructions(*aground_flag);
        while(TRUE)
        {
                /* read the instrument panel */
                read_settings(scan_mode, x_coord, y_coord, depth, course,
speed,
                        dive_angle, selected_course, selected_speed,
selected_dive_angle,
                        set, drift, cog, sog, tilt_inc, &done);
                if (done)
                        break;
                else
                {

                        /* indicate the position of the auv on the chart */
                        make_position_plus(*x_coord, *y_coord);

                        /* make the auv instrument readout */
                        make_readout(*scan_mode, *x_coord, *y_coord,
                        *depth, *course, *speed, *dive_angle,
                        *selected_course, *selected_speed,
                        *selected_dive_angle, *set, *drift, *cog, *sog,
                *tilt_inc);

                        /* make the depth indicator bar */
                        make_depth_bar(*x_coord, *y_coord, *depth);

                        /* make the instruction billboard */
                        make_instructions(*aground_flag);


                        swapbuffers();
                }

        } /* while */

        /* reset aground flag */
```

```
    *aground_flag = FALSE;

} /* get_auv_settings */
```

```
/*          +-----------------------------+
            |
            |      init_arrays.c
            |
            +-----------------------------+                */

/* Initializes the depth arrays and max/min values */

#include "sonar.h"
#include "gl.h"

initialize_depth_arrays(sweep_location,
   left_depth_array_active,
   left_array_max_depth, left_array_min_depth,
   right_array_max_depth, right_array_min_depth,
   x_coord, y_coord, depth, left_x_coord, left_y_coord, left_depth,
   right_x_coord, right_y_coord, right_depth)

   short *left_depth_array_active;
   int *sweep_location;
   int *left_array_max_depth, *left_array_min_depth,
         *right_array_max_depth, *right_array_min_depth;
   float x_coord, y_coord, depth,
         *left_x_coord, *left_y_coord, *left_depth,
         *right_x_coord, *right_y_coord, *right_depth;

{
   extern int
left_depth_array[SONAR_RESOLUTION][SONAR_RESOLUTION];
   extern int
right_depth_array[SONAR_RESOLUTION][SONAR_RESOLUTION];
   int i, j;

   for (i=0; i<SONAR_RESOLUTION; ++i)
   {
         for (j=0; j<SONAR_RESOLUTION; ++j)
         {
               right_depth_array[i][j] = 99999;
               left_depth_array[i][j] = 99999;
         }
```

```
    }

    *right_array_min_depth = 99999;
    *right_array_max_depth = -99999;
    *left_array_min_depth = 99999;
    *left_array_max_depth = -99999;
    *left_depth_array_active = TRUE;
    *sweep_location = 225;
    *left_x_coord = x_coord;
    *left_y_coord = -1 * y_coord;
    *left_depth = depth;
    *right_x_coord = x_coord;
    *right_y_coord = -1 * y_coord;
    *right_depth = depth;

} /* end initialize_depth_arrays */
```

```
/*                  +-----------------------------+
                    |
                    |       init_iris.c
                    |
                    +-----------------------------+                 */

    /* This procedure is called by main_sonar to initialize the
     graphics environment for the iris workstation  */

    #include "gl.h"             /* graphics definitions   */
    #include "device.h"           /* device definitions   */
    #include "sonar.h"             /* sonar constants  */

    init_iris()

    {


      static unsigned short cursordef[16] ={
0x3FF8,0x2108,0x2D68,0x2108,
            0x3D78,0x2548,0x2548,0x2448,
            0x27C8,0x2008,0xE00E,0xE00E,
            0xE00E,0x2008,0x2008,0x1FF0};

      int i; /* loop control */

    /* initialize the IRIS system */

    prefposition(632*XMAXSCREEN/1000,XMAXSCREEN,156*YMAX
SCREEN/1000,51*YMAXSCREEN/100);

    noborder();

    win_id_readout = winopen("readout");


      doublebuffer();

     /* put the IRIS into double buffer mode */
```

```
    gconfig();              /* (means use the above command settings) */

  prefposition(632*XMAXSCREEN/1000,XMAXSCREEN,511*YMAX
SCREEN/1000,YMAXSCREEN);


  noborder();
  win_id_array = winopen("array");
    doublebuffer();

   /* put the IRIS into double buffer mode */

    gconfig();              /* (means use the above command settings) */

  prefposition(0,XMAXSCREEN,0,YMAXSCREEN);
  win_id_wesmar = winopen("wesmar");




    doublebuffer();          /* put the IRIS into double buffer mode */
    gconfig();              /* (means use the above command settings) */



     /* exit key */
   qdevice(REDRAW);
   qdevice(INPUTCHANGE);
   qdevice(ESCKEY);
   qenter(REDRAW,0);
     qdevice(UPARROWKEY);
     qdevice(DOWNARROWKEY);
   qdevice(MENUBUTTON);
   qdevice(LEFTMOUSE);
   qdevice(MIDDLEMOUSE);

  /* define my cursor that looks like a sub  */
```

```
curstype(C16X1);
  defcursor(MY_CURSOR,cursordef);
drawmode(CURSORDRAW);
mapcolor(1,255,0,0);
drawmode(NORMALDRAW);
  setcursor(MY_CURSOR);



  /* set noise values */
  noise(MOUSEX,3);
  noise(MOUSEY,3);
  noise(DIAL0,2);
  noise(DIAL1,2);
  noise(DIAL2,2);
  noise(DIAL3,2);
  noise(DIAL4,6);
  noise(DIAL5,6);
  noise(DIAL6,6);

  /* set valuators */
  setvaluator(MOUSEX,0,0,XMAXSCREEN);
  setvaluator(MOUSEY,0,0,YMAXSCREEN);
  setvaluator(DIAL0,0,0,90);    /* power */
  setvaluator(DIAL1,0,0,360);   /* volume/hoist */
  setvaluator(DIAL2,0,0,280);   /* sector */
  setvaluator(DIAL3,0,0,300);   /* range */
  setvaluator(DIAL4,0,0,360);   /* course */
  setvaluator(DIAL5,0,0,360);   /* set */
  setvaluator(DIAL6,0,-90,90);  /* dive_angle */


  /* set up colors for sonar video */
  mapcolor(GREY,120,120,120);
  mapcolor(ORANGE,255,131,0);

  for (i=0; i<24; ++i)
        mapcolor(i+16,0,0,i*11);   /* dark blue to blue (16 - 39)*/

  for (i=0; i<24; ++i)
```

```
            mapcolor(i+40,i*11,i*11,255);   /* light blue to white (40-63)
*/

        /* colors for chartmask */
        for (i=64; i<128; ++i)
            mapcolor(i,0,0,0);   /* black (64-127) */

        for (i=128; i<256; ++i)
            mapcolor(i,255,0,0);   /* red (128-255) */

        for (i=256; i<512; ++i)
            mapcolor(i,0,255,0);   /* green (256-511) */

        for (i=512; i<1024; ++i)
            mapcolor(i,255,255,0);   /* yellow (512-1023) */

        for (i=1024; i<2048; ++i)
            mapcolor(i,255,0,255);   /* magenta (1024-2047)*/

        color(BLACK);
        clear();
        swapbuffers();

    } /* init_iris */
```

```
/*          +-----------------------------+
            |
            |    load_data.c
            |
            +-----------------------------+                  */

/*This module constructs the terrain data for the NPS swimming
pool*/

#include "sonar.h"

load_bottom_data()
{


  short row, col;   /* loop indicies  */
                                          extern          int
bottom_data[BOTTOM_POINTS_WIDTH][BOTTOM_POINTS_HEIGH
T];


float fcol;
  extern int max_depth, min_depth;

  /* read the data from the data file into the bottom_data array */



  for (col = 0; col < BOTTOM_POINTS_HEIGHT;++col)
    {
fcol = col;
    for (row = 0; row < BOTTOM_POINTS_WIDTH+1;++row)
          {

if(row<=2||row>=349)
{
        bottom_data[row][col] = 0;
}
if(row>2&&row<=156)
```

```
{
        bottom_data[row][col] = (int)( (.0088*row+1.062)*10);
}
if(row>156 && row<=175)
{
        bottom_data[row][col]= (.007693*row+1.2327)*10;
}
if(row>175&&row<349)
{
        bottom_data[row][col] = (-1.7391E-3*row+2.8956)*10;
}
if(col <= 2 || col >= 202)
{
        bottom_data[row][col] =0;
}

                /* check if depth is a max or a min */
        if (bottom_data[row][col] > max_depth)
                        max_depth = bottom_data[row][col];
                if (bottom_data[row][col] < min_depth)
                        min_depth = bottom_data[row][col];

/*      printf("%d\n",bottom_data[row][col]);*/
   }

  }
}/* load_bottom_data */
```

```
/*                    +-----------------------------+
                      |
                      |        make_arrows.c
                      |
                      +-----------------------------+                */

#include "sonar.h"
#include "math.h"
#include "gl.h"

make_arrows(x_coord, y_coord, depth, course, speed, set, drift,
   cog, sog, sweep_location, beam_length, tilt_angle, range_setting)

float x_coord, y_coord, depth, speed, sog, drift, beam_length;
int course, set, cog, tilt_angle, sweep_location, range_setting;
{

   /* varibles for location of arrow tips */
   static float course_arrow_x, course_arrow_y,
         current_arrow_x, current_arrow_y,
         cog_arrow_x, cog_arrow_y,
         l_course_x, l_course_y,
         r_course_x, r_course_y,
         l_current_x, l_current_y,
         r_current_x, r_current_y,
         l_cog_x, l_cog_y,
         r_cog_x, r_cog_y;

   /* convert y_coord to positive y */
   y_coord = -1 * y_coord;

   /* compute coords of arrow line segments */
   /* course arrow */
   course_arrow_x = x_coord + (ARROW_FACTOR * sin(course *
DTOR) * speed);
   course_arrow_y = y_coord + (ARROW_FACTOR * cos(course *
DTOR) * speed);
   l_course_x = course_arrow_x - (ARROW_WING_FACTOR *
         sin((course + 45) * DTOR) * speed);
   l_course_y = course_arrow_y - (ARROW_WING_FACTOR *
```

```
            cos((course + 45) * DTOR) * speed);
        r_course_x = course_arrow_x + (ARROW_WING_FACTOR *
            sin((course + 135) * DTOR) * speed);
        r_course_y = course_arrow_y + (ARROW_WING_FACTOR *
            cos((course + 135) * DTOR) * speed);

    /* current arrow */
        current_arrow_x = x_coord + (ARROW_FACTOR * sin(set *
DTOR) * drift);
        current_arrow_y = y_coord + (ARROW_FACTOR * cos(set *
DTOR) * drift);
        l_current_x = current_arrow_x - (ARROW_WING_FACTOR *
            sin((set + 45) * DTOR) * drift);
        l_current_y = current_arrow_y - (ARROW_WING_FACTOR *
            cos((set + 45) * DTOR) * drift);
        r_current_x = current_arrow_x + (ARROW_WING_FACTOR *
            sin((set + 135) * DTOR) * drift);
        r_current_y = current_arrow_y + (ARROW_WING_FACTOR *
            cos((set + 135) * DTOR) * drift);

        /* cog arrow */
        cog_arrow_x  = course_arrow_x + (current_arrow_x - x_coord);
        cog_arrow_y = course_arrow_y + (current_arrow_y - y_coord);
        l_cog_x = cog_arrow_x - (ARROW_WING_FACTOR *
            sin((cog + 45) * DTOR) * sog);
        l_cog_y = cog_arrow_y - (ARROW_WING_FACTOR *
            cos((cog + 45) * DTOR) * sog);
        r_cog_x = cog_arrow_x + (ARROW_WING_FACTOR *
            sin((cog + 135) * DTOR) * sog);
        r_cog_y = cog_arrow_y + (ARROW_WING_FACTOR *
            cos((cog + 135) * DTOR) * sog);



    viewport(0,632*XMAXSCREEN/1000,156*YMAXSCREEN/1000,Y
MAXSCREEN);
        ortho2(0.0,351.0,0.0,351.0);


        linewidth(2);
```

```
    /* make a red circle that shows the max beam range */
    /* course arrow */
    color(A_GREEN);
    move2(x_coord,y_coord);
    draw2(course_arrow_x, course_arrow_y);
    draw2(l_course_x, l_course_y);
    move2(course_arrow_x, course_arrow_y);
    draw2(r_course_x, r_course_y);

    /* current arrow */
    color(A_YELLOW);
    move2(x_coord, y_coord);
    draw2(current_arrow_x, current_arrow_y);
    draw2(l_current_x, l_current_y);
    move2(current_arrow_x, current_arrow_y);
    draw2(r_current_x, r_current_y);

    /* cog arrow */
    color(A_RED);
    move2(x_coord, y_coord);
    draw2(cog_arrow_x, cog_arrow_y);
    draw2(l_cog_x, l_cog_y);
    move2(cog_arrow_x, cog_arrow_y);
    draw2(r_cog_x, r_cog_y);

    /* put a red circle over the position */
    color(A_RED);
    circf(x_coord, y_coord,2);

    /* make a red circle that shows the max beam range */
    circ(x_coord, y_coord,(float)range_setting);
/*
    color(A_MAGENTA);
    pushmatrix();
    translate(x_coord, y_coord,0.0);
    rotate((sweep_location + course + 4) * -10, 'z');
    move2(0.0,0.0);
    draw2(beam_length * cos(tilt_angle * DTOR), 0.0);
    popmatrix();
```

```
*/
   /* make a red circle that shows the max beam range */
   circ(x_coord, y_coord,(float)range_setting/5);

} /* end make_arrows */
```

```
/*              +-----------------------------+
                |
                |       make_chart.c
                |
                +-----------------------------+                 */

/* make_chart.c - this procedure is called by main_sonar to
   build an object containing a chart map.     */

#include "gl.h"
#include "sonar.h"
#include "math.h"

make_chart()
{
   short i, j,l,m;
   float length, depth_float;
   int depth;
   /* vars used for color conversion from depth */
   Colorindex colour, lastcolor;
   e x t e r n                                          i n t
bottom_data[BOTTOM_POINTS_WIDTH][BOTTOM_POINTS_HEIGH
T]; /* terrain elevations */
   extern int max_depth, min_depth;


   viewport(0,632*XMAXSCREEN/1000,156*YMAXSCREEN/1000,
YMAXSCREEN);
        linewidth(14); /* for large chart (6 for small chart)*/
        ortho2(0.0,71.0,0.0,71.0);    /* use array index space */

        color(BLACK);
        clear();

        lastcolor = BLACK;

        for (i=0; i < 72; ++i)
        {       /* draw column i */
                move2(i + 0.5,0.0); /* start at bottom of column  */
```

```
                length = 0.0;    /* # adjacent points of the same color */
                for (j=0; j< 72;++j)
                {
                        l=i*5;
                        m=j*5;
                        depth = bottom_data[m][l];
                        depth_float = (float)depth;

                        /* convert depth to a color */
                        colour = (Colorindex)(MAX_COLOR_NUMBER
-
                                (((depth_float-min_depth)/(max_depth-
min_depth)) * NUMBER_OF_COLORS));

                        /* min color number is 16 */
                        if (colour < 16) colour = 16;

                        /* max color number is 63 */
                        if (colour > 63) colour = 63;


                        if (colour == lastcolor) length++;   /* don't draw
yet */
                        else
                        {                       /* draw now that color has
changed */
                                color(lastcolor);
                                rdr2(0.0,length);
                                lastcolor = colour;             /* reset for
new draw */
                                length = 1;
                        }
                } /* end for j */

                color(colour);                  /* draw last (top) line */
                rdr2(0.0,length);

        } /* end for i */

        linewidth(2);
```

```
        color(BLACK);                /* draw chart border */
        rect(0.0,0.0,80.0,80.0);

}  /* end make_chart */
```

```
/*          +-----------------------------+
            |
            |      make_depth.c
            |
            +-----------------------------+                */

/* make_depth_bar - this procedure is called by
   get_auv_settings to build the depth selection bar in the
   lower right hand corner of the screen */

#include "gl.h"
#include "sonar.h"

make_depth_bar(x_coord, y_coord, depth)
   float x_coord, y_coord, depth;

{
   int water_depth;
  float fwater_depth;
   char str[40];
   e x t e r n                                          i n t
bottom_data[BOTTOM_POINTS_WIDTH][BOTTOM_POINTS_HEIGH
T];

  /*convert from screen to pool coordinates*/


   /* calc water depth at current position */
   w a t e r _ d e p t h      =      b o t t o m _ d a t a [ ( i n t ) ( ( - 1      *
y_coord)/RESOLUTION_IN_METERS)][(int)(x_coord/RESOLUTION_I
N_METERS)];
/*printf("%f%d\n",x_coord,water_depth);*/

/* set up box */
viewport(632*XMAXSCREEN/1000,XMAXSCREEN,0,156*YMAX
SCREEN/1000);
   ortho2(0.0,100.0,0.0,33.0);
   color(CYAN);
   clear();
```

```
    color(BLACK);
    cmov2(39.0,29.0);
    charstr("0");
    cmov2(50.0,12.0);
    charstr("AUV DEPTH");
    cmov2(67.0,18.0);
    charstr("meters");

    /* print out auv depth */
    sprintf(str, "%5.1f", depth/10);
    cmov2(53.0,18.0);
    charstr(str);

    /* print out water depth on depth bar */
    sprintf(str, "%5.1f",(float)water_depth/10);
    cmov2(36.0,2.0);
    charstr(str);

    /* draw the depth bar */
    color(WHITE);
    rectf(27.0,3.0,33.0,30.0);
    color(BLUE);
    linewidth(22);
    if (depth != 0)
    {
            move2(30.0,30.0);
            draw2(30.0, 30.0 - ((27.0 * depth)/water_depth));
    }

    /* border of depth bar */
    linewidth(2);
    color(BLACK);
    rect(27.0,3.0,33.0,30.0);

} /* end make_depth_bar */
```

```
/*          +-----------------------------+
            |
            |      make_eplus.c
            |
            +-----------------------------+                */

/* make_position_plus - this procedure is called by
   main_sonar.c  to build the estimated position_plus on the large chart
   (the plus marks the estimated location of the auv)*/

#include "gl.h"
#include "sonar.h"

make_estimate_position_plus(x_coord, y_coord)
   float x_coord, y_coord;

{
   /* convert -y to pos y */
   y_coord = -1 * y_coord/5;
   x_coord = x_coord/5;

   viewport
(0,632*XMAXSCREEN/1000,156*YMAXSCREEN/1000,YMAXSCREE
N);
   ortho2(0.0,71.0,0.0,71.0);

   linewidth(4);
   color(A_MAGENTA);

   /* draw the plus over the position */
   move2(x_coord, y_coord);
   draw2(x_coord + PLUS_FACTOR, y_coord);
   move2(x_coord, y_coord);
   draw2(x_coord - PLUS_FACTOR, y_coord);
   move2(x_coord, y_coord);
   draw2(x_coord, y_coord + PLUS_FACTOR);
   move2(x_coord, y_coord);
   draw2(x_coord, y_coord - PLUS_FACTOR);
```

```
        /* decide where to put the "estimated position" label */
        if ((x_coord < 500) && (y_coord <= 500))
                cmov2(x_coord + PLUS_FACTOR, y_coord +
PLUS_FACTOR);
        else if ((x_coord < 500) && (y_coord > 500))
                cmov2(x_coord + PLUS_FACTOR, y_coord -
PLUS_FACTOR);
        else if ((x_coord >= 500) && (y_coord <= 500))
                cmov2(x_coord - (7 * PLUS_FACTOR), y_coord +
PLUS_FACTOR);
        else
                cmov2(x_coord - (7 * PLUS_FACTOR), y_coord -
PLUS_FACTOR);

        charstr("POSITION ESTIMATE");

    } /* end make_position_plus */
```

```
/*          +-----------------------------+
            |
            |      make_inst.c
            |
            +-----------------------------+                */

    /* make_instructions - this procedure is called by
       get_auv_settings to build the instruction billboard in the
       upper right hand corner of the screen */

    #include "gl.h"
    #include "sonar.h"

    make_instructions(aground_flag)
       short aground_flag;

    {
       short i; /* loop control */

       /* make exit sign at bottom */
       viewport
(0,632*XMAXSCREEN/1000,0,156*YMAXSCREEN/1000);
       ortho2(0.0,100.0,0.0,18.0);

       color(BLUE);
       clear();

       color(WHITE);

       if (aground_flag)
       {

            cmov2(7.0,10.0);
            charstr("WHEN ALL AUV ATTRIBUTES HAVE
DESIRED SETTINGS");
            cmov2(7.0,6.0);
            charstr("USE RIGHT MOUSE BUTTON TO CHANGE
SCREENS");
            /* draw aground flag */
            color(RED);
```

```
            rectf(82.0, 0.0,100.0,18.0);
            color(BLACK);
            cmov2(85.0,8.0);
            charstr("AGROUND!!");
        }
        else /* not aground */
        {
            cmov2(16.0,10.0);
            charstr("UPARROW & DOWNARROW KEYS
INCREMENT TILT");
            cmov2(16.0,6.0);
            charstr("USE RIGHT MOUSE BUTTON TO CHANGE
SCREENS");
        }

        /* dials/mouse sign in upper right hand corner */
        viewport
(632*XMAXSCREEN/1000,XMAXSCREEN,511*YMAXSCREEN/1000
,YMAXSCREEN);
        ortho2(0.0,100.0,0.0,100.0);

        color(MAGENTA);
        clear();

        color(WHITE);
        cmov2(12.0,90.0);
        charstr("SET AUV POSITION, COURSE & SPEED");
        cmov2(19.0,82.0);
        charstr("AND CURRENT SET & DRIFT");

        /* dial box */
        color(GREY);
        rectf(16.0,16.0,48.0,76.0);
        /* mouse box */
        rectf(60.0,40.0,84.0,72.0);
        /* dial box border, circles */
        linewidth(2);
        color(BLACK);
        rect(16.0,16.0,48.0,76.0);
        for (i=0; i< 4; ++i)
```

```
{
        circf(24.0, 23.0 + (i * 15), 4.0);
        circf(40.0, 23.0 + (i * 15), 4.0);
}
cmov2(18.0, 44.0);
charstr("COURSE");
cmov2(37.5, 44.0);
charstr("SET");
cmov2(20.0, 59.0);
charstr("DIVE");

/* mouse cord/ buttons */
/* cord */
rectf(71.0, 72.0,72.0, 76.0);
/* buttons */
for (i=0; i< 3; ++i)
{
        rectf(62.0 + (8 * i), 44.0,66.0 + (8 * i), 58.0);
}
/* mouse border */
rect(60.0,40.0,84.0,72.0);

/* writing at bottom of screen */
color(WHITE);
cmov2(23.0,7.0);
charstr("'DIALS'");

cmov2(82.0,31.0);
charstr("MENU");

cmov2(60.0,31.0);
charstr("SET SPEED,");
cmov2(60.0,27.0);
charstr("DRIFT AND");
cmov2(60.0,22.0);
charstr("AUV LOCATION");
cmov2(60.0,15.5);
charstr("'MOUSE'");

/* arrows */
```

```
    /* set arrow */
    move2(64.0, 36.0);
    draw2(64.0, 42.0);
    draw2(65.0, 40.5);
    move2(64.0, 42.0);
    draw2(63.0, 40.5);

    /* middle arrow not used right now */
    /*move2(82.0, 32.0);
    draw2(72.0, 42.0);
    draw2(72.0, 40.5);
    move2(72.0, 42.0);
    draw2(73.5, 42.0);*/

    /* menu arrow */
    move2(82.0, 36.0);
    draw2(82.0, 42.0);
    draw2(83.0, 40.5);
    move2(82.0, 42.0);
    draw2(81.0, 40.5);

} /* end make_instructions */
```

```
/*          +-----------------------------+
            |
            |      make_plus.c
            |
            +-----------------------------+                */

/* make_position_plus - this procedure is called by
   get_auv_settings to build the position_plus on the large chart
   on the left of the screen (the plus marks the location of the auv)*/

#include "gl.h"
#include "sonar.h"

make_position_plus(x_coord, y_coord)
   float x_coord, y_coord;

{
   /* convert -y to pos y */
   y_coord = -1 * y_coord/5;
   x_coord = x_coord/5;

   viewport
(0,632*XMAXSCREEN/1000,156*YMAXSCREEN/1000,YMAXSCREE
N);
   ortho2(0.0,351.0,0.0,351.0);

make_chart();
   linewidth(4);
   color(A_RED);

   /* draw the plus over the position */
   move2(x_coord, y_coord);
   draw2(x_coord + PLUS_FACTOR, y_coord);
   move2(x_coord, y_coord);
   draw2(x_coord - PLUS_FACTOR, y_coord);
   move2(x_coord, y_coord);
   draw2(x_coord, y_coord + PLUS_FACTOR);
   move2(x_coord, y_coord);
   draw2(x_coord, y_coord - PLUS_FACTOR);
```

```
        /* decide where to put the "current position" label */
        if ((x_coord < 500) && (y_coord <= 500))
                cmov2(x_coord + PLUS_FACTOR, y_coord +
PLUS_FACTOR);
        else if ((x_coord < 500) && (y_coord > 500))
                cmov2(x_coord + PLUS_FACTOR, y_coord -
PLUS_FACTOR);
        else if ((x_coord >= 500) && (y_coord <= 500))
                cmov2(x_coord - (7 * PLUS_FACTOR), y_coord +
PLUS_FACTOR);
        else
                cmov2(x_coord - (7 * PLUS_FACTOR), y_coord -
PLUS_FACTOR);

        charstr("CURRENT POSITION");

    } /* end make_position_plus */
```

```
/*          +-----------------------------+
            |
            |       make_read.c
            |
            +-----------------------------+                 */

/* make_readout - this procedure is called by main_sonar and
   get_auv_settings to build the auv instrument readout on the
   right hand corner of the screen */

#include "gl.h"
#include "sonar.h"

make_readout(scan_mode, x_coord, y_coord, depth, course, speed,
   dive_angle, selected_course, selected_speed, selected_dive_angle,
   set, drift, cog, sog, tilt_inc)

   float x_coord, y_coord, depth, speed, drift, sog, selected_speed;
   int course, set, dive_angle, cog, selected_course,
         selected_dive_angle, tilt_inc;
   short scan_mode;

{
   char str[40];
   short i; /* loop control */
   winset((long) win_id_readout);
   winpop();
   ortho2(0.0,100.0,0.0,74.0);

   color(BLACK);
   clear();

   /* course and speed */
   color(GREEN);
   rectf(0.0,37.5,66.0,74.0);
   color(WHITE);
   circf(13.0,62.0,9.0);
   /* speed bar*/
   rectf(29.75,53.0,36.25,71.0);
   /* course box*/
```

```
rectf(8.0,45.0,18.0,50.0);
/* speed box*/
rectf(28.0,45.0,38.0,50.0);
/* dive angle arc */
arcf(44.0,62.0,9.0,2700,900);
/* dive box */
rectf(48.0,45.0,58.0,50.0);

/* indicator titles */
color(BLACK);
cmov2(6.0,40.0);
charstr("COURSE");
cmov2(26.0,40.0);
charstr("SPEED");
cmov2(42.0,40.0);
charstr("DIVE  ANGLE");
/* labels on speed bar */
/* high mark */
/*sprintf(str, "%5.1f", MAX_SPEED);*/
sprintf(str, "%2d", 12);
cmov2(24.0,70.0);
charstr(str);
/* low mark */
/*sprintf(str, "%5.1f", MIN_SPEED);*/
sprintf(str, "%d", -4);
cmov2(24.0,52.0);
charstr(str);
/* zero mark */
linewidth(1);
move2(29.0, ZERO_Y);
draw2(36.25, ZERO_Y);
cmov2(25.0, ZERO_Y - 1.0);
charstr("0");

/* dive angle labels */
cmov2(55.0, 68.0);
charstr("UP");
cmov2(56.0, 61.0);
charstr("0");
cmov2(55.0, 53.0);
```

```
charstr("DOWN");

linewidth(2);
/* outline course, speed and dive angle arcs and boxes */
/* course circle */
circ(13.0,62.0,9.0);
/* course box */
rect(8.0,45.0,18.0,50.0);
/* speed box */
rect(28.0,45.0,38.0,50.0);
/* speed bar */
rect(29.75,53.0,36.25,71.0);
/* dive angle arc */
move2(44.0,71.0);
draw2(44.0, 53.0);
arc(44.0, 62.0,9.0,2700,900);
rect(48.0,45.0,58.0,50.0);

/* put the numbers in the boxes */
/* course */
sprintf(str, "%3d", course);
cmov2(9.0,46.0);
charstr(str);
/* degree circle */
circ(16.5,48.5,0.50);

/* speed */
sprintf(str, "%3.1f", speed);
cmov2(28.7,46.0);
charstr(str);

/* dive angle */
sprintf(str, "%3d", dive_angle);
cmov2(49.0,46.0);
charstr(str);
/* degree circle */
circ(56.5,48.5,0.50);

/* draw in speed bar */
linewidth(24);
```

```
if (selected_speed != 0)
{
        move2(33.0, ZERO_Y);
        draw2(33.0, ZERO_Y + (selected_speed * SPEED_INC));
}
linewidth(2);

/* draw in course indicator */
pushmatrix();
translate(13.0, 62.0,0.0);
rotate(selected_course * -10, 'z');
translate(-13.0,-62.0,0.0);
move2(13.0,62.0);
draw2(13.0, 71.0);
popmatrix();

/* draw in dive_angle indicator */
pushmatrix();
translate(44.0, 62.0,0.0);
rotate(selected_dive_angle * -10, 'z');
translate(-44.0,-62.0,0.0);
move2(44.0,62.0);
draw2(53.0,62.0);
popmatrix();

/* now display current info */
color(YELLOW);
rectf(0.0,0.0,66.0,37.0);
color(WHITE);
circf(17.0,25.0,9.0);
/* drift bar */
rectf(46.0,16.0,53.0,34.0);
/* set box */
rectf(11.0,8.0,24.0,13.0);
/* drift box */
rectf(43.0,8.0,57.0,13.0);
color(BLACK);
/* outline boxes */
/* set box */
rect(11.0,8.0,24.0,13.0);
```

```
/* drift box */
rect(43.0,8.0,57.0,13.0);
/* set circle border*/
circ(17.0,25.0,9.0);


cmov2(13.0,3.0);
charstr("SET");
cmov2(43.0,3.0);
charstr("DRIFT");
sprintf(str, "%d", MAX_DRIFT);
cmov2(43.0,33.0);
charstr(str);
cmov2(43.0,15.0);
charstr("0");

/* set readout */
sprintf(str, "%3d", set);
cmov2(13.0,9.0);
charstr(str);
/* degree circle */
circ(22.5,11.5,0.50);

/* drift readout */
sprintf(str, "%3.1f", drift);
cmov2(46.0,9.0);
charstr(str);

/* drift bar */
linewidth(24);
if (drift != 0)
{
        move2(49.5,16.0);
        draw2(49.5,16.0 + (DRIFT_INC * drift));
}
linewidth(2);
rect(46.0,16.0,53.0,34.0);

/* set circle indicator */
pushmatrix();
```

```
translate(17.0,25.0,0.0);
rotate(set * -10, 'z');
translate(-17.0, -25.0,0.0);
move2(17.0,25.0);
draw2(17.0,34.0);
popmatrix();

/* draw the number readout now */
/* AUV position */
color(BLUE);
rectf(66.0,38.0,100.0,74.0);
/* AUV cog sog */
color(RED);
rectf(66.0,14.0,100.0,38.0);
/* scan mode and tilt_inc */
color(MAGENTA);
rectf(66.0,0.0,100.0,14.0);
color(BLACK);
for (i=0; i< 4; ++i)
{
        move2(66.0, (i*12) + 26.0);
        draw2(100.0, (i*12) + 26.0);
}
move2(66.0, 14.0);
draw2(100.0, 14.0);

/* frame the readout */
rect(66.0,0.0,100.0,74.0);
/* seperate the AUV display from the ocean CURRENT display */
move2(0.0,37.0);
draw2(66.0,37.0);

color(WHITE);
/* number boxes */
for (i=0; i< 5; ++i)
{
        rectf(78.0,20.0 + (i * 12), 90.0, 25.0 + (i * 12));
}

/* enter numbers */
```

```
color(BLACK);
sprintf(str, "%5.1f", x_coord);
cmov2(78.5,69.0);
charstr(str);
sprintf(str, "%5.1f", y_coord);
cmov2(78.5,57.0);
charstr(str);
sprintf(str, "%5.1f", percent_noise*100);
cmov2(78.5,45.0);
charstr(str);
sprintf(str, "%3.1f", sog);
cmov2(81.0,21.0);
charstr(str);
sprintf(str, "%3d", cog);
cmov2(79.0,33.0);
charstr(str);
circ(89.0,36.0,0.50);

/* now label boxes */
color(WHITE);
cmov2(76.0,64.0);
charstr("X-COORD");
cmov2(76.0,52.0);
charstr("Y-COORD");
cmov2(76.0,40.0);
charstr("% Noise");
cmov2(80.0,28.0);
charstr("COG");
cmov2(74.0,16.0);
charstr("SOG (kts)");

color(BLACK);
cmov2(70.0,8.0);
charstr("SCAN MODE:");
cmov2(94.5,8.0);
if (scan_mode == JUST_SCAN_NO_STORE)
        charstr("0");
else if (scan_mode == ONE_SCAN_AND_STORE)
        charstr("1");
else
```

```
        charstr("2");
    cmov2(70.0,3.0);
    charstr("TILT INC:");
    sprintf(str, "%2d", tilt_inc);
    cmov2(92.0,3.0);
    charstr(str);

    swapbuffers();
    winset((long)win_id_wesmar);

}  /* end make_readout */
```

```
/*          +-----------------------------+
            |
            |     make_video.c
            |
            +-----------------------------+                */

/* make_sonar - this procedure is called by main_sonar
to simulate the AUV sonar . */

#include "gl.h"
#include "sonar.h"
#include "math.h"

make_sonar_video(range_setting, tilt_angle,
power_on, x_coord, y_coord, depth,
course, dive_angle, roll_angle, sweep_location, sweep_clockwise,
 beam_length, beam_inc, encountered_contact)


    int range_setting;
    int tilt_angle;
    short power_on;
    float x_coord, y_coord, depth, *beam_length, *beam_inc;
    int course, dive_angle, roll_angle, *sweep_location;
    short sweep_clockwise; /* boolean for sweep direction */
    short *encountered_contact;

    {
    e x t e r n                                          i n t
bottom_data[BOTTOM_POINTS_WIDTH][BOTTOM_POINTS_HEIGH
T];
    int depth_at_sweep_point;
    short hun_digit, tens_digit, ones_digit;
    int pos_tilt; /* negative tilt converted to positive number */
    int depth_value,bcourse;
    short i; /* loop control */
    static short sonar_running = FALSE;

    float c4,c5,c6,c7,c8,s4,s5,s6,s7,s8; /* vars for sin/cos of euler angles
*/
```

```
        short boundary_flag; /* boolean for database boundary hit */
        int  depth_at_tip, arc_tag, arc_index;
        int bottom_depth_at_tip(); /* function to return db depth value at
beam tip point */
        float find_bottom_inc(), /* increase beam function */
                find_bottom_dec(), /* decrease beam function */
                tip_depth(); /* depth of beam tip function */

        /* convert dive_angle for use in D-H matricies */
        /* dive_angle down is positive from readout, but negative for D-H */
        dive_angle = -1 * dive_angle;



        if (power_on)
        {
                        sonar_running = TRUE;

                        boundary_flag = FALSE; /* always set boundary flag
false to start */

    *beam_inc = 1;
    *beam_length = 0;
    /* calculate which video arc is to be updated */
                                if (*sweep_location == 0)
                                        arc_index = 35;
                                else if (*sweep_location == 10)
                                        arc_index = 0;
                                else
                                        arc_index = (*sweep_location / 10) - 1;
                        /* calculate sins and cos of Euler angles */

                        trig_calcs(course, dive_angle, roll_angle, tilt_angle,
                                *sweep_location, &c4, &c5, &c6, &c7, &c8,
&s4, &s5, &s6, &s7, &s8);

                        /* check to see whether to increment or decrement
beam_range */
                        depth_at_tip = bottom_depth_at_tip(x_coord, y_coord,
c4, c5, c6, c7, c8, s4, s5, s6, s7, s8, *beam_length);
```

```
                if (depth_at_tip == BOUNDARY_FLAG_VALUE)
                { /* at a boundary so reset to 0 and increment beam
length */
                        *beam_length = 0.0;
                        * b e a m _ l e n g t h                =
find_bottom_inc(&boundary_flag, range_setting,
                                *beam_inc, x_coord, y_coord, depth, c4,
c5, c6, c7, c8, s4, s5,
                                s6, s7, s8, *beam_length) - (0.5 *
*beam_inc);
                }
                else if (depth_at_tip >= tip_depth(depth, c4, c5, c6, c7,
c8, s4, s5,
                        s6, s7, s8, *beam_length))
                { /* sonar beam tip is above the bottom so increment */
                        * b e a m _ l e n g t h                =
find_bottom_inc(&boundary_flag, range_setting,
                                *beam_inc, x_coord, y_coord, depth, c4,
c5, c6, c7, c8, s4, s5,
                                s6, s7, s8, *beam_length) - (0.5 *
*beam_inc);
                }
                else
                { /* sonar beam tip is past the bottom so decrement
beam length */
                        *beam_length = find_bottom_dec(*beam_inc,
x_coord, y_coord, depth,
                                c4, c5, c6, c7, c8, s4, s5 ,s6, s7, s8,
*beam_length);
                        /* sonar beam tip is above the bottom so
increment */
                        * b e a m _ l e n g t h                =
find_bottom_inc(&boundary_flag, range_setting,
                                *beam_inc, x_coord, y_coord, depth, c4,
c5, c6, c7, c8, s4, s5,
                                s6, s7, s8, *beam_length) - (0.5 *
*beam_inc);
                }
```

```
                if (*beam_length > range_setting)
                {
                        arc_tag = 15; /* target arc is outside of sonar
screen */
                        *beam_length = range_setting; /* reset
beam_length */
                }
                else
                {
                        /* convert range to one of 15 arc tags (0 - 14)*/
                        arc_tag = ((*beam_length/range_setting) *
NUMBER_OF_ARCS);
                        *beam_length = *beam_length - (0.5 *
*beam_inc); /* reset beam_length */
                }


                if (boundary_flag)
                { /* database boundary reached before tip reached
bottom */

                }
                else  /* no boundary flag */
                {
                        if (arc_tag < NUMBER_OF_ARCS) /* contact
with bottom made */
                        {
                                *encountered_contact = TRUE;


                        }

                } /* else no boundary */



        } /* power on */
```

```
    bcourse = 90 + *sweep_location;
    if (bcourse > 360)
            {bcourse = bcourse -360;
            }
    if (bcourse < 0)
            {bcourse = bcourse + 360;
            }

    printf("\n Relative beam angle = %d,beam length = %f
meters",bcourse,*beam_length/10);


    } /* end make_video.c */


    /* function to find the row and col indices depending on x and y
       location of the sub and extract the value from the depth data base
*/
    int bottom_depth_at_tip(x_coord, y_coord, c4, c5, c6,
            c7, c8, s4, s5, s6, s7, s8, beam_length)

    float x_coord, y_coord, c4, c5, c6 ,c7, c8, s4, s5, s6, s7, s8,
beam_length; {
        extern int
bottom_data[BOTTOM_POINTS_WIDTH][BOTTOM_POINTS_HEIGH
T];
        int row_index, col_index;
        float a09_2_4(), a09_1_4();
        int value;

        row_index = (int)(-1 * a09_2_4(y_coord, c4, c5, c6, c7, c8, s4, s5,
s6,
            s7, s8, beam_length)/RESOLUTION_IN_METERS);
        col_index = (int)(a09_1_4(x_coord, c4, c5, c6, c7, c8, s4, s5, s6,
            s7, s8, beam_length)/RESOLUTION_IN_METERS);

        if ((row_index < 0) || (row_index > BOTTOM_POINTS_HEIGHT-
1) ||
```

```
        (col_index < 0) || (col_index > BOTTOM_POINTS_WIDTH-
1))
    {  /* out of database bounds */
        return BOUNDARY_FLAG_VALUE;
    }
    else
    {
        value = bottom_data[row_index][col_index];
        return value;
    }

} /* end bottom_depth_at_tip  */


/* function to increment the beam length until it hits bottom or
boundary */
float find_bottom_inc(boundary_flag, range_setting, beam_inc,
  x_coord, y_coord, depth, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8,
  beam_length)

float beam_inc, x_coord, y_coord,
 depth, c4, c5, c6 ,c7, c8,
  s4, s5, s6, s7, s8, beam_length;
short *boundary_flag;
int range_setting;

{
  int depth_at_tip;
  int bottom_depth_at_tip();
  float tip_depth();
  do
  {
        beam_length = beam_length + beam_inc;

        depth_at_tip = bottom_depth_at_tip(x_coord, y_coord, c4, c5,
c6,
              c7, c8, s4, s5, s6, s7, s8, beam_length);

  } while ((depth_at_tip > tip_depth(depth, c4, c5, c6, c7, c8, s4, s5,
```

```
            s6, s7, s8, beam_length)) && (beam_length <= range_setting)
&&
            (depth_at_tip != BOUNDARY_FLAG_VALUE));

      if ((depth_at_tip == BOUNDARY_FLAG_VALUE) || (beam_length
> range_setting))
            *boundary_flag = TRUE;

      return beam_length;
   /*printf("%f,%d\n",beam_length,depth_at_tip);*/
   }  /* end find_bottom_inc  */


   /* function to decrement the beam length until tip depth is less than
bottom */
   float find_bottom_dec(beam_inc, x_coord, y_coord, depth, c4, c5, c6,
      c7, c8, s4, s5, s6, s7, s8, beam_length)

   float beam_inc, x_coord, y_coord,
    depth, c4, c5, c6 ,c7, c8,
      s4, s5, s6, s7, s8, beam_length;

   {
      int bottom_depth_at_tip();       float tip_depth();


    do
      {
            beam_length = beam_length - beam_inc;
      } while (tip_depth(depth, c4, c5, c6, c7, c8, s4, s5,
            s6, s7, s8, beam_length) > bottom_depth_at_tip(x_coord,
y_coord,
            c4, c5, c6, c7, c8, s4, s5, s6, s7, s8, beam_length));

      return beam_length;
   }  /* end find_bottom_dec  */


   /* calculate the depth of the beam tip */
   float tip_depth(depth, c4, c5, c6, c7, c8, s4, s5,
```

```
        s6, s7, s8, beam_length)

    float depth, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8, beam_length;

    {
    float a09_3_4();
    float value;

      value = FEET_PER_M * a09_3_4(depth, c4, c5, c6, c7, c8, s4, s5,
s6, s7, s8, beam_length);
      return value;
    } /* end tip_depth  */


    /* calculate the sins/cos of Euler angles */
    trig_calcs(course, sub_dive_angle, sub_roll, tilt_angle,
      sweep_location, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8)

    float *c4, *c5, *c6, *c7, *c8, *s4, *s5, *s6, *s7, *s8;
    int course, sub_dive_angle, sub_roll, tilt_angle,
      sweep_location;

    {
      *c4 = cos (DTOR * course);
      *s4 = sin (DTOR * course);
      *c5 = cos (DTOR * sub_dive_angle);
      *s5 = sin (DTOR * sub_dive_angle);
      *c6 = cos (DTOR * sub_roll);
      *s6 = sin (DTOR * sub_roll);
      *c7 = cos (DTOR * tilt_angle);
      *s7 = sin (DTOR * tilt_angle);
      *c8 = cos (DTOR * sweep_location);
      *s8 = sin (DTOR * sweep_location);
    } /* end trig_calcs */


    /* x_coord of beam tip in base coord system */
    float a09_1_4(x_coord, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8,
beam_length)
```

```
    float x_coord, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8, beam_length;

    {
      float value;

            return (x_coord
                  + (c4 * c5 * beam_length * c7 * c8)
                  + ((beam_length * c7 * s8) * ((c4 * s5 * s6) - (s4 *
c6)))
                  - ((beam_length * s7) * ((c4 * s5 * c6) + (s4 * s6))));

    }  /* end a09_1_4 */


    /* y_coord of beam tip in base coord system */
    float a09_2_4(y_coord, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8,
beam_length)

    float y_coord, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8, beam_length;

    {
      float value;

            return (y_coord
                  + (s4 * c5 * beam_length * c7 * c8)
                  + ((beam_length * c7 * s8)*((c4 * c6) + (s4 * s5 *
s6)))
                  - ((beam_length * s7) * ((s4 * s5 * c6) - (c4 * s6))));

    }  /* end a09_2_4 */


    /* depth of beam tip in base coord system */
    float a09_3_4(depth, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8, beam_length)

    float depth, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8, beam_length;

    {
      float value;
```

```
        return (depth
                - (s5 * beam_length * c7 * c8)
                + (c5 * s6 * beam_length * c7 * s8)
                - (c5 * c6 * beam_length * s7));

} /* end a09_3_4 */
```

```
/*          +-----------------------------+
            |
            |      obs_filter.c
            |
            +-----------------------------+                */

/* makes positional observation and correct position estimate */

#include "sonar.h"
#include "gl.h"
#include "math.h"
#include "device.h"

observation_and_filter(scan_mode, drift, tilt_inc,
    tilt_angle, scan_complete, request_depth_array_display,
    range_setting, beam_inc, left_depth_array_active,
    left_array_max_depth,      left_array_min_depth,
right_array_max_depth,
    right_array_min_depth, left_x_coord, left_y_coord, left_depth,
    right_x_coord, right_y_coord, right_depth, auv_depth,
    estx_coord, esty_coord)
```

```
    short scan_complete, request_depth_array_display,
            left_depth_array_active, scan_mode;
    int range_setting, tilt_inc, tilt_angle;
    int left_array_max_depth, left_array_min_depth,
            right_array_max_depth, right_array_min_depth;
    float beam_inc, drift;
    float left_x_coord, left_y_coord, left_depth,
            right_x_coord, right_y_coord, right_depth, auv_depth,
            *estx_coord,*esty_coord;


{
    extern int bottom_data[BOTTOM_POINTS_HEIGHT][BOTTOM_POINTS_WIDTH];
    extern int left_depth_array[SONAR_RESOLUTION][SONAR_RESOLUTION];
    extern int right_depth_array[SONAR_RESOLUTION][SONAR_RESOLUTION];
    extern int max_depth, min_depth;
    char str[40];
```

```
    int i,j,row,col,r,c,row_start,col_start,bottom_value,k,rval,min_rval,cval,min_cval
;

    int array_depth, e_cog, a_cog, water_depth_at_ave;

    float cog_x, cog_y, a_distance, e_distance, ave_x, ave_y,

    depth_below_auv,zestx_coord,zesty_coord,num,x_check,y_check;

    Colorindex colour;

    short first_pass, too_many_passes, new_min_criterion,

        analysis_completed, first_match;

    int number_of_cells, number_of_passes, row_offset, col_offset,

        cell_location, row_inc, col_inc,row_change,col_change;

    float min_criterion, x_dist, y_dist;

    short right_array_empty, left_array_empty;

  int hititem;

    short value;


    if (scan_complete)

    {
        first_pass = TRUE;
```

```
x_check = 0;
y_check = 0;



        while (TRUE)

        {


    /*conduct observation*/


    if(left_depth_array_active)
    {
                                /* calc actual cog and distance traveled */
                                cog_x = left_x_coord - right_x_coord;
                                cog_y = left_y_coord - right_y_coord;
                                a_distance = sqrt((cog_x * cog_x)+(cog_y
* cog_y));
                                /* cog */
                                /* 9 cases */
                                if ((cog_x == 0) && (cog_y == 0))
                                        a_cog= 0;
                                else if ((cog_x == 0) && (cog_y > 0))
```

```
                    a_cog= 0;
                else if ((cog_x == 0) && (cog_y < 0))
                    a_cog= 180;
                else if ((cog_y == 0) && (cog_x < 0))
                    a_cog= 270;
                else if ((cog_y == 0) && (cog_x > 0))
                    a_cog= 90;
                else if ((cog_x < 0) && (cog_y > 0)) /*
270 - 360 */
                    a_cog= 270 + (int)(atan(cog_y/(-1 *
cog_x)) * RTOD);
                else if ((cog_x < 0) && (cog_y < 0)) /*
180 - 270 */
                    a_cog= 270 -
(int)(atan(cog_y/cog_x) * RTOD);
                else if ((cog_x > 0) && (cog_y > 0)) /* 0 -
90 */
                    a_cog= 90 - (int)(atan(cog_y/cog_x)
* RTOD);
                else if ((cog_x > 0) && (cog_y < 0)) /* 90
- 180 */
                    a_cog= 90 + (int)(atan((-1 *
cog_y)/cog_x) * RTOD);
```

```
} /* end if left active */
else /* right array active */
{ /* right array has new scan info */

        /* calc cog and distance traveled */
        cog_x = right_x_coord - left_x_coord;
        cog_y = right_y_coord - left_y_coord;
        a_distance = sqrt((cog_x *
        cog_x)+(cog_y * cog_y));
        /* cog */
        /* 9 cases */
        if ((cog_x == 0) && (cog_y == 0))
                a_cog = 0;
        else if ((cog_x == 0) && (cog_y > 0))
                a_cog = 0;
        else if ((cog_x == 0) && (cog_y < 0))
                a_cog = 180;
        else if ((cog_y == 0) && (cog_x < 0))
```

```
                a_cog = 270;

            else if ((cog_y == 0) && (cog_x > 0))

                a_cog = 90;

            else if ((cog_x < 0) && (cog_y > 0)) /*
            270 - 360 */

                a_cog = 270 + (int)(atan(cog_y/(-
            1 * cog_x)) * RTOD);

            else if ((cog_x < 0) && (cog_y < 0)) /*
            180 - 270 */

                a_cog = 270 -
            (int)(atan(cog_y/cog_x) * RTOD);

            else if ((cog_x > 0) && (cog_y > 0)) /* 0
            - 90 */

            a_cog = 90 - (int)(atan(cog_y/cog_x) *
            RTOD);

            else if ((cog_x > 0) && (cog_y < 0)) /*
            90 - 180 */

                a_cog = 90 + (int)(atan((-1 *
        cog_y)/cog_x) * RTOD);



        } /* end if right active */



        if (first_pass)
```

```
{
        first_match =  TRUE;
        number_of_cells = 0;
        number_of_passes = 0;
        too_many_passes = FALSE;
        min_criterion = 0.0;
        row_offset = 0;
        col_offset = 0;
        row_inc = 0;
        col_inc = 0;
        row_change = 0;
        col_change = 0;
        min_rval = 0;
        min_cval = 0;
        rval = 0;
        cval = 0;
        analysis_completed = FALSE;
              new_min_criterion = FALSE;
        printf("\n\nNEW COMPARE
        ************");
```

```
                    for (r = 0; r < CRIT_SIZER; r = r+1)

                    {
                        for(c = 0;c < CRIT_SIZEC; c = c+1)
                    {
                        /* clear all array cells */

                        criterion_array[r][c] = 0;
                    }
                    }


                        /* calculate criterion function for
                starting location */

*esty_coord = -*esty_coord;

if(*esty_coord < 0)
{*esty_coord = -*esty_coord;
}

zesty_coord = *esty_coord;
zestx_coord = *estx_coord;
        for (k = 1; k < 5; ++k)

        {


        col_start = (int)zestx_coord-150;

        row_start = (int) zesty_coord-150;
        row = row_start + depth_array[k][2];
        col = col_start + depth_array[k][3];

printf("depth%d,xcart%d,ycart%d,row%d,col%d\n",depth_array[k][1],
        depth_array[k][3],depth_array[k][2],row,col);
        if(row<0 || col<0 || row>351 || col >204)
{       bottom_value = -99999;
```

```
}
else
{

        bottom_value = (int)bottom_data[row][col] - (auv_depth +
        1) ;
        if(bottom_value < 0)
{
        bottom_value = 0;
}
}

        if ((depth_array[k][1] != 99999) &&(bottom_value !=
  99999))

                { /* both cells have data */

                number_of_cells = number_of_cells + 1;

min_criterion = min_criterion +((depth_array[k][1] -bottom_value )
                * (depth_array[k][1] -bottom_value ));

                                        } /* if data */


                        } /* outer for */


                        if (number_of_cells == 0)

                        { /* no need to continue */

                                first_match =  FALSE;

                                printf("\n\n NO CELLS MATCH
ON                      FIRST ANALYSIS");

                        }
```

```
        else

        { /* continue */

        /* normalize the criterion value with
        the # of cells used */
        if (min_criterion == 0)
        {min_criterion = 0;
        }
        else
        {
        min_criterion = min_criterion /
        number_of_cells;
        }
        printf("\n\n Starting min_criterion =
%f",min_criterion);

        printf("\n Starting number of cells =
%d",number_of_cells);



        /* now begin searching for best
                possible match of both

                depth arrays */



        while (TRUE)

        {



        printf("\n\n min_criterion =
%f",min_criterion);
```

```
                    /* start a new pass, increment
                    pass counter */

                    number_of_passes =
                    number_of_passes + 1;

                    printf("\n\n NEW PASS
                    PPPPPPPPPPP");

                    printf("\n\n Pass number =
                    %d",number_of_passes);


                    /* commence calculation of
                criterion function

                        cells in order */

                    for(r=0;r< CRIT_SIZER;r=r+1)
                    {
                    for(c=0;c< CRIT_SIZEC;c=c+1)
                    {
                    row_offset = -CRIT_SIZER/2 + r;
                    col_offset = -CRIT_SIZEC/2 +c;

/* we now have new cell_location (row/col position) */

/* calculate criterion function for the new cell */


        number_of_cells = 0;

        criterion_array[r][c] = 0;


for (k=1; k<5; ++k)
```

```
{

        col_start = (int) zestx_coord - 150;
        row_start = (int) zesty_coord -150;
        row = row_start + depth_array[k][2]-row_offset;
        col = col_start + depth_array[k][3]-col_offset;
        if(row<0 || col<0 || row>351 || col >204)
{       bottom_value = -99999;
}

        if(row>0 && row<351 && col>0 && col<204)

{
        bottom_value = (int)bottom_data[row][col]-
        (auv_depth+1);
        if(bottom_value < 0)
{
        bottom_value = 0;
}
}
        if((depth_array[k][1] != 99999) &&(bottom_value !=
  99999))

                        { /* both cells have data */

  number_of_cells = number_of_cells + 1;

  criterion_array[r][c] = criterion_array[r][c] +
  ((depth_array[k][1] -bottom_value) *(depth_array[k][1] -
  bottom_value));

} /*end data*/

                                        } /* end k for */
```

```
                                        if (number_of_cells == 0)
{
                                criterion_array[r][c] =MAX_INTEGER;
}
                                else
                        { /* normalize that criterion function value */
                                                criterion_array[r][c] =

   criterion_array[r][c]/number_of_cells;
                                                }
if(criterion_array[r][c] < 0)
{
criterion_array[r][c] = - criterion_array[r][c];
}
rval = (CRIT_SIZER/2 - r)*(CRIT_SIZER/2 - r);
cval = (CRIT_SIZEC/2 -c)*(CRIT_SIZEC/2 - c);
rval = (int) sqrt((double) rval);
cval = (int) sqrt((double) cval);

        if(criterion_array[r][c] < min_criterion)

        {
                min_criterion = criterion_array[r][c];
                new_min_criterion = TRUE;
                row_inc = row_offset;
                col_inc = col_offset;
                min_rval = rval;
                min_cval = cval;
        }/*end if new min criterion*/

/*To make sure min_criterion is as close to the center as possible*/
```

```
    if(min_criterion == criterion_array[r][c] && (min_rval > rval ||
min_cval > cval))

    {
                row_inc = row_offset;
                col_inc = col_offset;
                min_rval = rval;
                min_cval = cval;
    }       /* end value closest to center */



                                } /* end for r */
                                    }/*end for c*/



                                /* now see if new min_criterion is
not center cell */

                                if (new_min_criterion)

                                { /* we moved */
                        row_change = row_change +row_inc;
                        col_change = col_change +col_inc;

                        zesty_coord = zesty_coord - row_inc;

                        zestx_coord = zestx_coord - col_inc;

                        row_inc = 0;
                        col_inc = 0;
                        new_min_criterion = FALSE;

                        if (number_of_passes > MAX_PASSES)

                                        {

                            too_many_passes = TRUE;
```

```
                    analysis_completed = TRUE;
                    new_min_criterion = FALSE;
                        }
                    } /* end if new min */
                else /* at best match - no movement */
                    {
                        analysis_completed = TRUE;
                        new_min_criterion = FALSE;
                    }

                    /* print out info */
                    if (too_many_passes)
        printf("\n\n REACHED MAXIMUM NUMBER OF PASSES!");

        printf("\n\n row_offset = %d", row_change);
        printf("\n col_offset = %d", col_change);

        /* determine distance moved */
        y_dist = row_change * beam_inc * -1;
        x_dist = col_change * beam_inc * -1;
```

```
printf("\n\n x_dist = %f", x_dist);
printf("\n y_dist = %f", y_dist);

        /* calc estimated dog & cog */
        /* 9 cases */
        if ((x_dist == 0) && (y_dist == 0))
        e_cog= 0;
        else if ((x_dist == 0) && (y_dist > 0))
         e_cog= 0;
        else if ((x_dist == 0) && (y_dist < 0))
        e_cog= 180;
        else if ((y_dist == 0) && (x_dist < 0))
        e_cog= 270;
        else if ((y_dist == 0) && (x_dist > 0))
        e_cog= 90;
else if ((x_dist < 0) && (y_dist > 0)) /* 270 - 360 */
e_cog= 270 + (int)(atan(y_dist/(-1 * x_dist)) * RTOD);
else if ((x_dist < 0) && (y_dist < 0)) /* 180 - 270 */
e_cog= 270 - (int)(atan(y_dist/x_dist) * RTOD);
else if ((x_dist > 0) && (y_dist > 0)) /* 0 - 90 */
```

```
e_cog= 90 - (int)(atan(y_dist/x_dist) * RTOD);
else if ((x_dist > 0) && (y_dist < 0)) /* 90 - 180 */
e_cog= 90 + (int)(atan((-1 * y_dist)/x_dist) * RTOD);
printf("\n\n ESTIMATED COG = %d", e_cog);

/* distance over ground */
e_distance = sqrt((x_dist * x_dist) + (y_dist * y_dist));
printf("\n ESTIMATED DISTANCE = %f", e_distance/10);

        if (analysis_completed)
                break;

                } /* end while TRUE */

        printf("\n\nCOMPARE COMPLETE #########");
printf("\n\nnumber of moves = %d", number_of_passes - 1);
printf("\n\n ACTUAL COG = %d degrees", a_cog);
printf("\n ESTIMATED SET = %d", e_cog);
printf("\n\n ACTUAL DISTANCE = %f meters", a_distance/10);
```

```
        printf("\n FINAL ESTIMATED DISTANCE = %f",
e_distance/10);

        /* calc number of moves horizontally and vertically */

                                /* actual verticle/ horizontal */
        printf("\n\n Total horiz. grid displacement = %d",
        (int)(sin(a_cog * DTOR) * a_distance / beam_inc));
        printf("\n Total vert. grid displacement = %d",
        (int)   (cos(a_cog * DTOR) * a_distance / beam_inc));

        /* estimated verticle/ horizontal */
        printf("\n\n Estimated horiz. grid displacement = %d",
        (int)(x_dist / beam_inc));
        printf("\n Estimated vert. grid displacement = %d",
                                        (int)(y_dist / beam_inc));

                                if (left_depth_array_active)
                                {

            printf("\n Actual x_coord = %f", left_x_coord);
            printf("\n Actual y_coord = %f", left_y_coord);
```

```
                    }

                    else /* right active */

                    {

printf("\n Actual x_coord = %f", right_x_coord);

printf("\n Actual y_coord = %f", right_y_coord);

                    }

                    r11 = min_criterion;

                    r33 = min_criterion;
                    if(min_criterion != 0)
                    {
                     r11 = min_criterion/30;
                     r33 = min_criterion/1000;
                    }
                    if(number_of_cells == 1)
                    {
                    r11 = 1000;
                    if(r33 < 10)
                    {
                     r33 = 100;
                    }
                    }
                    if(number_of_cells == 2 &&
          min_criterion < 2)
                    {

                     r11 = 5;
                     r33 = 3;
                    }


                    if (r11 < 3)
                    {
```

```
                    r11 = 3;
                    }


        /*Check to see if there is degraded x or y sonar info
                                because of return geometry*/

                    if(number_of_cells > 2)

                    {
                     for(k=1;k<number_of_cells;++k)

                     {
          x_check = x_check + depth_array[k][2];


                     y_check = y_check + depth_array[k][3];


        }
             x_check = x_check/(number_of_cells - 1);
             y_check = y_check/(number_of_cells - 1);

if( (x_check>(float)depth_array[1][2]- 20.0)&&
(x_check<(float) depth_array[1][2] + 20.0))
                    {
                     r11 = 1000.0;
                    }

if (y_check > ((float)depth_array[1][3] -  20.0)
&& y_check < ((float)depth_array[1][3] + 20.0))
                    {
                     r33 = 1000.0;
                    }

                    x_check = 0;

                    y_check = 0;

                    }/*end if # of cells greater than 2*/
```

```
/* reduce observation weight if observation is outside of
                        the charted area*/

                    if(zesty_coord > 349 || zesty_coord < 3)
                    {
                        r33 = 10000;
                    }

                    if(zestx_coord > 202 || zestx_coord < 3)
                    {
                        r11 = 10000;
                    }


                    g11 = p11*(1/(p11+r11));


                    g33 = p33*(1/(p33+r33));
printf("\n Starting Estimated x_coord =%f",*estx_coord);
printf("\n Starting Estimated y_coord =%f",*esty_coord);

printf("\n Uncorrected Estimated x_coord =
%f",zestx_coord);

printf("\n Uncorrected Estimated y_coord =
%f",zesty_coord);
*estx_coord = *estx_coord +g11*(zestx_coord -
*estx_coord);

*esty_coord = *esty_coord +g33*(zesty_coord -
*esty_coord);

                    p11 = (1 - g11)*p11;

                    p33 = (1 - g33)*p33;

                printf("\n X gain =%f Y gain = %f",g11,g33);
```

```
                    q11 = .50;

                    q33 = 1.0;

                    p11 = p11 + q11;
                    p33 = p33 + q33;
                    printf("\n   Estimated   x_coord   =
%f",*estx_coord);
                    printf("\n   Estimated   y_coord   =
%f",*esty_coord);

                    printf("\n Drift = %f knots", drift);

    printf("\n Range Setting = %d meters", range_setting/10);

        switch (scan_mode)
            {
        case ONE_SCAN_AND _STORE:
        printf("\n Scan Mode = ONE SCAN");
        printf("\n Tilt Angle = %d degrees", tilt_angle);
        break;
```

```
        case COMPLETE_SCAN_AND_STORE:
        printf("\n Scan Mode = COMPLETE SCAN");
        printf("\n Tilt Increment = %d degrees\n\n", tilt_inc);
        break;

                        }

                        } /* end else continue, first match =
                        TRUE */

                        first_pass = FALSE;

                    } /* end if first pass */

*esty_coord = -*esty_coord;
break;
                } /* end else not exit */

        } /* end while TRUE */
} /* end obs filter */
```

```
/*          +-----------------------------+
            |
            |       read_ctrls.c
            |
            +-----------------------------+                */

/* This procedure is called by main_sonar to read the values
   from the operator's controls (mouse, dials and keyboard) */

#include "gl.h"                 /* graphics lib defs */
#include "sonar.h"                /* sonar constants */
#include "device.h"             /* device definitions */

read_controls(scan_mode,

   tilt_angle_change,
   power_on, hoist_down,
   tilt_angle,
   tilt_inc,
   exit, selected_course, selected_speed,
   selected_dive_angle, set, drift, change_location,
```

```
    stop)

    short *scan_mode,
          *tilt_angle_change,*power_on, *hoist_down,
          *exit, *change_location;
          int *tilt_angle,
          *tilt_inc,*selected_course,
          *selected_dive_angle, *set,*stop;
    float *selected_speed, *drift;

{
 int mainmenu;
 int hititem;

    extern int
bottom_data[BOTTOM_POINTS_HEIGHT][BOTTOM_POINTS_WIDT
H];
    float x,y, y_ortho,oldx,oldy;

    int inside();
```

```
short value;

*power_on = TRUE;
*hoist_down = TRUE;

if (qtest())

        {

        switch(qread(&value))

        {

        case MENUBUTTON:

        if(value==1)

        {hititem=dopup(mainmenu_ctrls);

        processmenu(hititem,scan_mode,stop,change_location,exit);

        break;

        }
```

```
 case UPARROWKEY:
    if(value==1)
    {
        *tilt_inc = *tilt_inc + 1;
        if (*tilt_inc > 90)
        {       *tilt_inc = 90;}
        qreset();
            break;
    }

case DOWNARROWKEY:
    if(value==1)
    {
        *tilt_inc = *tilt_inc - 1;
        if (*tilt_inc < 0)
    {           *tilt_inc = 0;}
            qreset();
            break;
    }
    /* if left mouse button hit, check for pick box */
```

```
case LEFTMOUSE:  /* read x and y from mouse */
if(value==1)
{
        x = getvaluator(MOUSEX)*1023/XMAXSCREEN;
        y = getvaluator(MOUSEY)*767/YMAXSCREEN;



        /* do we have a speed bar hit? */
        if (inside(x,y,29.75,36.25,53.0,71.0,0.0,100.0,0.0,74.0,
                647,1023,120,391)) /* hit on speed bar */
        {
                /* convert y to ortho coords */
                /* y_ortho = ((y - vminy)*(omaxy -
ominy))/(vmaxy - vminy); */
                y_ortho = ((y-120)*74.0)/271;
                /* ortho bar range = omaxy - ominy = 18 */
                /* convert y_ortho to auv speed */
                /* *selected_speed = ((MAX_SPEED -
MIN_SPEED) * (y_ortho - ZERO_Y))/ortho bar range */
                *selected_speed = ((MAX_SPEED -
MIN_SPEED) * (y_ortho - ZERO_Y))/18;
```

```
                }

                /* do we have a drift bar hit? */
                if (inside(x,y,46.0,53.0,16.0,34.0,0.0,100.0,0.0,74.0,
                        647,1023,120,391)) /* hit on drift bar */
                {
                        /* convert y to ortho coords */
                        /* y_ortho = ((y - vminy)*(omaxy -
ominy))/(vmaxy - vminy) */
                        y_ortho = ((y-120)*74.0)/271;

                        /* ortho bar range = omaxy - ominy = 18 */
                        /* convert y_ortho to current drift */
                        /* *drift = ((y_ortho - ominy)*
MAX_DRIFT)/ortho_bar_range*/
                        *drift = ((y_ortho - 16)*MAX_DRIFT)/18;
                }

        break;

        } /* LEFTMOUSE */
```

```
            default:

            break;

            }


    }

            else{

            if (getvaluator(DIAL4) != *selected_course) /* change in
course dial */

            {

                    *selected_course = getvaluator(DIAL4);

            }


            if (getvaluator(DIAL5) != *set) /* change in set dial */

            {

                    *set = getvaluator(DIAL5);

            }


            if (getvaluator(DIAL6) != *selected_dive_angle) /*change in
dive_angle dial*/

            {
```

```
                *selected_dive_angle = getvaluator(DIAL6);
        }


        /* if MOUSE2, increment to toggle values selected */



    } /* not exit */

/*freepup(mainmenu);*/

} /* read_controls */
processmenu(hititem,scan_mode,stop,change_location,exit)
int hititem,*stop;
short *scan_mode,*change_location,*exit;

{
switch(hititem)
{
case -1:  /*No selection*/
```

```
break;
case 1:
*scan_mode = JUST_SCAN_NO_STORE;
break;

case 2:
*scan_mode = ONE_SCAN_AND_STORE;
break;
case 3:
*scan_mode = COMPLETE_SCAN_AND_STORE;
break;
case 4:

*stop = 0;
*change_location = TRUE;
break;
case 5:
*stop = 0;
break;
case 6:
```

```
go = 0;
break;
case 7:
*stop = 0;
go = 1;
break;
case 8:
qreset();
*stop = 0;
*exit = TRUE;              /* quit */
break;
} /*switch*/
return;
} /*processmenu*/
```

```
/*          +-----------------------------+
            |
            |       read_ctrls.c
            |
            +-----------------------------+                */

/* This procedure is called by main_sonar to read the values
   from the operator's controls (mouse, dials and keyboard) */

#include "gl.h"                 /* graphics lib defs */
#include "sonar.h"                /* sonar constants */
#include "device.h"             /* device definitions */

read_controls(scan_mode,

   tilt_angle_change,
   power_on, hoist_down,
   tilt_angle,
   tilt_inc,
   exit, selected_course, selected_speed,
   selected_dive_angle, set, drift, change_location,
```

```
    stop)

    short *scan_mode,
          *tilt_angle_change,*power_on, *hoist_down,
          *exit, *change_location;
          int *tilt_angle,
          *tilt_inc,*selected_course,
          *selected_dive_angle, *set,*stop;
    float *selected_speed, *drift;

{
 int mainmenu;
 int hititem;

    extern int
bottom_data[BOTTOM_POINTS_HEIGHT][BOTTOM_POINTS_WIDTH];

    float x,y, y_ortho,oldx,oldy;

    int inside();
```

```
short value;

*power_on = TRUE;
*hoist_down = TRUE;

if (qtest())
        {
        switch(qread(&value))
        {
        case  MENUBUTTON:
        if(value==1)
        {hititem=dopup(mainmenu_ctrls);
        processmenu(hititem,scan_mode,stop,change_location,exit);
        break;
        }
         case UPARROWKEY:
        if(value==1)
        {
                *tilt_inc = *tilt_inc + 1;
                if (*tilt_inc > 90)
                {       *tilt_inc = 90;}
                qreset();
```

```
                    break;
        }

case DOWNARROWKEY:
        if(value==1)
        {
                *tilt_inc = *tilt_inc - 1;
                if (*tilt_inc < 0)
        {               *tilt_inc = 0;}
                        qreset();
                        break;
        }
        /* if left mouse button hit, check for pick box */
        case LEFTMOUSE:  /* read x and y from mouse */
        if(value==1)
        {
                x = getvaluator(MOUSEX)*1023/XMAXSCREEN;
                y = getvaluator(MOUSEY)*767/YMAXSCREEN;
                /* do we have a speed bar hit? */
                if (inside(x,y,29.75,36.25,53.0,71.0,0.0,100.0,0.0,74.0,
```

```
                    647,1023,120,391)) /* hit on speed bar */

                {

                    /* convert y to ortho coords */

                    /* y_ortho = ((y - vminy)*(omaxy -
ominy))/(vmaxy - vminy); */

                    y_ortho = ((y-120)*74.0)/271;

                    /* ortho bar range = omaxy - ominy = 18 */

                    /* convert y_ortho to auv speed */

                    /* *selected_speed = ((MAX_SPEED -
MIN_SPEED) * (y_ortho - ZERO_Y))/ortho bar range */

                    *selected_speed = ((MAX_SPEED -
MIN_SPEED) * (y_ortho - ZERO_Y))/18;

                }
                /* do we have a drift bar hit? */

                if (inside(x,y,46.0,53.0,16.0,34.0,0.0,100.0,0.0,74.0,

                    647,1023,120,391)) /* hit on drift bar */

                {

                    /* convert y to ortho coords */

                    /* y_ortho = ((y - vminy)*(omaxy -
ominy))/(vmaxy - vminy) */

                    y_ortho = ((y-120)*74.0)/271;
                    /* ortho bar range = omaxy - ominy = 18 */

                    /* convert y_ortho to current drift */
```

```
                        /*  *drift  =  ((y_ortho  -  ominy)*
MAX_DRIFT)/ortho_bar_range*/

                        *drift = ((y_ortho - 16)*MAX_DRIFT)/18;

                }
        break;
        } /* LEFTMOUSE */

        default:
        break;

        }



    }

        else{
        if (getvaluator(DIAL4) != *selected_course) /* change in
course dial */

        {

                *selected_course = getvaluator(DIAL4);

        }
        if (getvaluator(DIAL5) != *set) /* change in set dial */

        {

                *set = getvaluator(DIAL5);

        }
        if (getvaluator(DIAL6) != *selected_dive_angle) /*change in
dive_angle dial*/

        {

                *selected_dive_angle = getvaluator(DIAL6);
```

```
            }
            /* if MOUSE2, increment to toggle values selected */
    } /* not exit */
/*freepup(mainmenu);*/


} /* read_controls */

processmenu(hititem,scan_mode,stop,change_location,exit)

int hititem,*stop;

short *scan_mode,*change_location,*exit;
{

switch(hititem)

{

case -1:  /*No selection*/

break;

case 1:

*scan_mode = JUST_SCAN_NO_STORE;

break;
case 2:

*scan_mode = ONE_SCAN_AND_STORE;

break;

case 3:

*scan_mode = COMPLETE_SCAN_AND_STORE;

break;
```

```
case 4:

*stop = 0;
*change_location = TRUE;
break;
case 5:
*stop = 0;
break;
case 6:
go = 0;
break;
case 7:
*stop = 0;
go = 1;
break;
case 8:
q-eset();
*stop = 0;
*exit = TRUE;              /* quit */
break;
```

```
} /*switch*/
return;
} /*processmenu*/
```

```
/*          +-----------------------------+
            |
            |      reset_depth_array.c
            |
            +-----------------------------+                 */

/* reset_depth_array.c - this procedure is called by main_sonar to
   reset all the change flags */

#include "gl.h"
#include "sonar.h"

reset_depth_array()


{
 int i,j;
for(i=0;i<8;i++)
{
for(j=0;j<4;j++)
{
depth_array[i][j] = 99999;
}/*inner for*/
}/*outer for*/
arg = 1;
} /* reset_depth_array */
```

```
/*          +-----------------------------+
            |
            |     reset_flags.c
            |
            +-----------------------------+                  */

/* reset_flags - this procedure is called by main_sonar to
   reset all the change flags */

#include "gl.h"
#include "sonar.h"

reset_flags(scan_complete,sector_setting_change,
   encountered_contact)

   short *scan_complete,*sector_setting_change,
         *encountered_contact;

{
   *scan_complete = FALSE;
   *sector_setting_change = FALSE;
   *encountered_contact = FALSE;

} /* reset_flags */
```

```
/*          +-----------------------------+
            |
            |       scan_ctrl.c
            |
            +-----------------------------+                 */

/* This procedure is called by main_sonar to determine
   when a scan has completed depending upon the scan mode */

#include "sonar.h"
#include "gl.h"

bottom_scan_controller(scan_mode, power_on, hoist_down,
   tilt_angle, tilt_angle_change, tilt_inc, sweep_location,
   scan_complete)

   short scan_mode, power_on, hoist_down,
         *tilt_angle_change, *scan_complete;
   int *tilt_angle, tilt_inc, sweep_location;

{
         if (power_on && hoist_down && (sweep_location == 225))
   { /* sonar is operational and is at the end of a 360 deg. sweep*/
/* determine scan mode */
               if (scan_mode == ONE_SCAN_AND_STORE)
                     *scan_complete = TRUE;
               else
               { /* scan mode = COMPLETE_SCAN_AND_STORE
*/
                     /* decrement tilt_angle */
                     *tilt_angle = *tilt_angle - tilt_inc;
                     *tilt_angle_change = TRUE;
                     if (*tilt_angle < -90)
                     {
                           *scan_complete = TRUE;
                           *tilt_angle = 0; /* reset to 0 */
                     }
               } /* end else */
         } /* end if */
} /* end bottom_scan_controller */
```

```
/*          +-----------------------------+
            |
            |    store_data.c
            |
            +-----------------------------+                */

    /* Stores the information received from the sonar in an array
       so regression analysis may be performed. */

    #include "sonar.h"
    #include "math.h"
    #include "gl.h"

    store_return_data(scan_mode, encountered_contact, auv_depth,
       sweep_location,beam_length, beam_inc, tilt_angle,
       left_depth_array_active,     left_array_max_depth,
left_array_min_depth,
       right_array_max_depth, right_array_min_depth,course)

       short scan_mode, encountered_contact, left_depth_array_active;
       int sweep_location, tilt_angle,course;
       float auv_depth, beam_length, beam_inc;

       int *left_array_max_depth, *left_array_min_depth,
              *right_array_max_depth, *right_array_min_depth;

    {
     int j_factor; /*added to beam length to measure depth beyond tip*/
       int x_cart, y_cart, depth_cart,idum; /* cartesian conversion    vars
*/
       int x_index, y_index; /* depth array indicies from cartesian
       coords */
       extern                                                  int
left_depth_array[SONAR_RESOLUTION][SONAR_RESOLUTION];
       extern                                                  int
right_depth_array[SONAR_RESOLUTION][SONAR_RESOLUTION];
       extern int min_depth, max_depth;
    float gasdev(),ran1();
    float junk;
    idum = 1;
```

```
junk = 0;
   if (scan_mode != JUST_SCAN_NO_STORE)
   { /* must be one of other two scans */
if (-tilt_angle > 42)
{j_factor = 0;

}
else
{j_factor = 2;
}
        if (encountered_contact)
        {  /* convert the contact location from polar to cart.
        coords */
              junk = gasdev(&idum);/*random noise generator*/
              printf("\n noise = %f",junk);
              beam_length = beam_length
        +(percent_noise*beam_length*junk);
              x_cart = (int)((sin((sweep_location+course) *
        DTOR) * cos(tilt_angle * DTOR)*
        (beam_length+j_factor)));
              y_cart = (int)((cos((sweep_location+course) *
        DTOR) * cos(tilt_angle * DTOR) *
        (beam_length+j_factor)));
  depth_cart = (int)(((sin(-tilt_angle * DTOR) * beam_length)));


              /* check for depth calculations that are greater
        than max depth or less than the min depth of the
        database */
              if (depth_cart > max_depth)
                    depth_cart = max_depth;
              else if (depth_cart <= min_depth)
                    depth_cart = min_depth;

              /* now convert these cartesian coords into array
              indicies */
              /* on 300 by 300 grid, (0 - 290) 150 - 150 is the
        center */

              x_index = 150 - x_cart;
```

```
y_index = 150 + y_cart;

if ((x_index < 300) && (y_index < 300))
{ /* indicies are within array bounds */

        /* we now have the location of the contact in
        array indicies*/
        /* load the depth of the contact into the
array using these
                indicies */

        if (left_depth_array_active)
        {  /* load into left depth array */
                left_depth_array[x_index][y_index] =
        depth_cart;
        if (depth_cart >*left_array_max_depth)
        *left_array_max_depth = depth_cart;

        else if (depth_cart < *left_array_min_depth)
        *left_array_min_depth= depth_cart;
        }
        else /* right array active */
        {
        right_depth_array[x_index][y_index] =
        depth_cart;
        if (depth_cart > *right_array_max_depth)
        *right_array_max_depth = depth_cart;
        else if (depth_cart <*right_array_min_depth)
        *right_array_min_depth = depth_cart;
        }

/*load depth_array for numerical calculations*/


depth_array[arg][1] = depth_cart;
depth_array[arg][2] = x_index;
depth_array[arg][3] = y_index;
arg = arg+1;

} /* end if indicies in bounds */
```

```
            } /* if encountered contact */

        } /* if not JUST_SCAN */

    } /* end store_return_data */

    /*Routine to generate random numbers to provide gaussian sonar
noise*/
    /*This Routine taken from numerical recipies in C p216*/

    float  gasdev(idum)
    int *idum;
    {
                    static int iset = 0;
                    static float gset;
                    float fac,r,v1,v2;
                    float ran1();

    if (iset == 0) {
                            do{
                                    v1=2.0*ran1(idum)-1.0;
                                    v2=2.0*ran1(idum)-1.0;
                                    r=v1*v1+v2*v2;
                                    } while (r >= 1.0);
                                    fac=sqrt(-2.0*log(r)/r);
                                    gset=v1*fac;
                                    iset=1;
                                    return v2*fac;
            }else{
                                    iset=0;
                                    return gset;
            }
    }

    /*this routine computes random numbers*/

    #define M1 259200
    #define IA1 7141
    #define IC1 54773
```

```
#define RM1 (1.0/M1)
#define M2 134456
#define IA2 8121
#define IC2 28411
#define RM2 (1.0/M2)
#define M3 243000
#define IA3 4561
#define IC3 51349

float ran1(idum)
int *idum;

{
        static long ix1,ix2,ix3;
        static float r[98];
        float temp;
        static int iff=0;
        int j;
        if(*idum < 0 || iff == 0)
{
                iff=1;
                ix1=(IC1-(*idum))%M1;
                ix1=(IA1*ix1+IC1) %M1;
                ix2=ix1 % M2;
                ix1=(IA1*ix1+IC1) %M1;
                ix3=ix1 % M3;
                for(j=1;j<=97;j++)
{
                        ix1=(IA1*ix1+IC1) % M1;
                        ix2=(IA2*ix2+IC2) % M2;
                        r[j]=(ix1+ix2*RM2)*RM1;
        }
        *idum=1;
}
ix1=(IA1*ix1+IC1) % M1;
ix2=(IA2*ix2+IC2) % M2;
ix3=1 + (IA3*ix3+IC3) % M3;
j=1 + ((97*ix3)/M3);
if(j>97||j<1)
{printf("RAN1:This cannot Happen. ");
```

```
exit(1);
}
temp = r[j];
r[j]=(ix1+ix2*RM2)*RM1;
return temp;
}
```

```
/*          +-----------------------------+
            |
            |      up_arrays.c
            |
            +-----------------------------+                  */

/* Toggles depth arrays and reinitializes as necessary */

#include "sonar.h"
#include "gl.h"

update_depth_arrays(scan_complete, x_coord, y_coord, depth,
    left_depth_array_active, left_array_max_depth,
    left_array_min_depth, right_array_max_depth,
right_array_min_depth,
    left_x_coord, left_y_coord, left_depth, right_x_coord, right_y_coord,
    right_depth)

    short *left_depth_array_active, scan_complete;
    int *left_array_max_depth, *left_array_min_depth,
        *right_array_max_depth, *right_array_min_depth;
    float x_coord, y_coord, depth,
        *left_x_coord, *left_y_coord, *left_depth,
        *right_x_coord, *right_y_coord, *right_depth;

{
    extern int
left_depth_array[SONAR_RESOLUTION][SONAR_RESOLUTION];
    extern int
right_depth_array[SONAR_RESOLUTION][SONAR_RESOLUTION];
    int i, j;

    if (scan_complete)
    { /* a full sweep has been completed */
        /* reinitialize inactive array */
        if (*left_depth_array_active)
        { /* switch to right_active and reinit right */
            *left_depth_array_active = FALSE;
            *right_array_min_depth = 99999;
            *right_array_max_depth = -99999;
```

```
                /* save AUV position with right array */
                *right_x_coord = x_coord;
                *right_y_coord = -1 * y_coord;
                *right_depth = depth;

                for (i=0; i<SONAR_RESOLUTION; ++i)
                {
                        for (j=0; j<SONAR_RESOLUTION; ++j)
                        {
                                right_depth_array[i][j] = 99999;
                        }
                }
        } /* if left active */
        else /* right array active */
        {  /* switch to left active, reinit left */
                *left_depth_array_active = TRUE;
                *left_array_min_depth = 99999;
                *left_array_max_depth = -99999;

                /* save AUV position with left array */
                *left_x_coord = x_coord;
                *left_y_coord = -1 * y_coord;
                *left_depth = depth;

                for (i=0; i<SONAR_RESOLUTION; ++i)
                {
                        for (j=0; j<SONAR_RESOLUTION; ++j)
                        {
                                left_depth_array[i][j] = 99999;
                        }
                }

        } /* if right active */

   } /* if scan_complete */

} /* end update_depth_arrays */
```

```
/* Makefile*/
CFLAGS = -lgl -lm -g
OBJS = disp_arrays.o \
disp_estimate.o \
control_sweep.o \
get_estimate.o\
get_posit.o \
get_sets.o \
init_arrays.o \
init_iris.o \
load_data.o \
main_sonar.o \
make_arrows.o \
make_chart.o \
make_depth.o \
make_eplus.o \
make_inst.o \
make_plus.o \
make_read.o \
make_sonar.o \
obs_filter.o \
read_ctrls.o \
read_sets.o \
reset_depth_array.o\
reset_flags.o \
scan_ctrl.o \
store_data.o \
up_arrays.o

HDRS = disp_arrays.o \
disp_estimate.o \
control_sweep.o \
get_estimate.o\
get_posit.o \
get_sets.o \
init_arrays.o \
init_iris.o \
load_data.o \
main_sonar.o \
make_arrows.o \
```

```
make_chart.o \
make_depth.o \
make_eplus.o \
make_inst.o \
make_plus.o \
make_read.o \
make_sonar.o \
obs_filter.o \
read_ctrls.o \
read_sets.o \
reset_depth_array.o\
reset_flags.o \
scan_ctrl.o \
store_data.o \
up_arrays.o

nav: $(OBJS)
   cc -o nav $(OBJS) $(CFLAGS)

$(HDRS): sonar.h
```

## LIST OF REFERENCES


1. Robinson, R. C., "National Defense Applications of Autonomous Underwater Vehicles," IEEE *Journal of Oceanic Engineering*, v. OE-11, No. 4, pp. 462-467, October 1986.

2. Programmable Sonar Altimeter Reference Manual, The Model PSA-900, Datasonics Inc., Cataumet, Massachusetts.

3. Putnam, R. G. "A Conceptual Design of an Internal Navigation System for an Autonomous Submersible Testbed Vehicle", M. S. Thesis, Naval Postgraduate School, Monterey California, September 1987.

4 Easton, J. K. Jr., "Estimating Forces Acting on an Underwater Vehicle with G. P. S. and Kalman Filtering", M. S. Thesis, Naval Postgraduate School, Monterey California, March 1988.

5. Hartley, C. A., "A Computer Simulation Study of Station Keeping by an Autonomous Submersible using Bottom-Tracking Sonar", M. S. Thesis, Naval Postgraduate School, Monterey California, June 1988.

6. Schwartz, M. A., "Kalman Filtering for Adaptive Depth, Steering and Roll Control of an Autonomous Underwater Vehicle (AUV)", M. S. Thesis, Naval Postgraduate School, Monterey California, June 1989.

7. Rickenbach, M. D., "Correction of Inertial Navigation System Errors for an Autonomous Land Vehicle Using Optical Terrain Data", M. S. Thesis, Naval Postgraduate School, Monterey California, September 1987.

8. Titus, H. A., Notes for EC3310 (Optimal Control), Naval Postgraduate School, Monterey California, 1989.

## INITIAL DISTRIBUTION LIST

| | | No. Copies |
|---|---|---|
| 1. | Defense Technical Information Center<br>Cameron Station<br>Alexandria, Virginia 22304-6145 | 2 |
| 2. | Library, Code 0142<br>Naval Postgraduate School<br>Monterey, California 93943-5100 | 2 |
| 3. | Chairman, Code 62<br>Department of Electrical and Computer Engineering<br>Monterey, California 93943-5004 | 1 |
| 4. | Department of Electrical and Computer Engineering<br>Naval Postgraduate School<br>Monterey, California 93943-5004<br>ATTN: Professor R. Cristi, Code 62Cx | 8 |
| 5. | Department of Mechanical Engineering<br>Naval Postgraduate School<br>Monterey, California 93943-5004 | 2 |
| 6. | Department of Computer Science<br>Naval Postgraduate School<br>Monterey, California 93943-5004 | 2 |
| 7. | Department of Computer Science<br>Naval Postgraduate School<br>Monterey, California 93943-5004<br>ATTN: Professor M. Zyda, Code 52Zk | 1 |
| 8. | Commander, Naval Surface Weapons Center<br>White Oak, Maryland 20910<br>ATTN: R. Werneth, Code U25 | 1 |
| 9. | Head, Undersea AI and Robotics Branch<br>Naval Ocean System Center<br>San Diego, California 92152<br>ATTN: P. Heckman, Code 943 | 1 |

10. Commander, Naval Sea Systems Command (PMS-350) 1
Washington, D.C. 20362-5101
ATTN: Ms Judy Rumsey

11. Commander, Naval Coastal Systems Center 1
Panama City, Florida 32407-5000
ATTN: Ms Judy Rumsey

12. David Taylor Naval Costal Systems Center 1
Carderock, Laboratory
Bethesda, Maryland 20084-5000
ATTN: Dr. D. Milne, Code 1563

12. Commanding Officer 1
Polaris Missile Facility Atlantic
Charleston, South Carolina 29408-5700
ATTN: LCDR John Friend